AD A271 768

J. Komorowski Z.W. Raś (Eds.)

Methodologies for Intelligent Systems

7th International Symposium, ISMIS '93
Trondheim, Norway, June 15-18, 1993
Proceedings

Accesion For
NTIS
DTIC
Unannounced
Justification
By per th
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

DTIC QUALITY INSPECTED 3

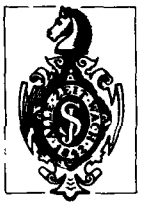
Best
Available
Copy

Lecture Notes in Artificial Intelligence 689

Subseries of Lecture Notes in Computer Science
Edited by J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos and J. Hartmanis

N00014-93-1-0244



Series Editor

Jörg Siekmann
University of Saarland
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, W-6600 Saarbrücken 11, FRG

Volume Editors

Jan Komorowski
Knowledge Systems Group
Faculty of Computer Science and Electrical Engineering
The Norwegian Institute of Technology, The University of Trondheim
N-7034 Trondheim, Norway

Zbigniew W. Raś
Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA

CR Subject Classification (1991): I.2

ISBN 3-540-56804-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56804-2 Springer-Verlag New York Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1993
Printed in Germany

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

This volume contains papers which were selected for presentation at the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS'93, held in Trondheim, Norway, June 15-18, 1993. The symposium was hosted by the Norwegian Institute of Technology and sponsored by The University of Trondheim, NFR/NTNF – The Norwegian Research Council, UNC-Charlotte, Office of Naval Research, Oak Ridge National Laboratory and ESPRIT BRA Compulog Network of Excellence.

ISMIS is a conference series that was started in 1986 in Knoxville, Tennessee. It has since then been held in Charlotte, North Carolina, once in Knoxville, and once in Torino, Italy.

The Organizing Committee has decided to select the following major areas for ISMIS'93:

- Approximate Reasoning
- Constraint Programming
- Expert Systems
- Intelligent Databases
- Knowledge Representation
- Learning and Adaptive Systems
- Manufacturing
- Methodologies

The contributed papers were selected from more than 120 full draft papers by the following Program Committee: Jens Balchen (NTH, Norway), Alan W. Biermann (Duke, USA), Alan Bundy (Edinburgh, Scotland), Jacques Calmet (Karlsruhe, Germany), Jaime Carbonell (Carnegie-Mellon, USA), David Hislop (US Army Research Office), Eero Hyvonen (VTT, Finland), Marek Karpinski (Bonn, Germany), Yves Kodratoff (Paris VI, France), Jan Komorowski (NTH, Norway), Kurt Konolige (SRI International, USA), Catherine Lassez (Yorktown Heights, USA), Lennart Ljung (Linköping, Sweden), Ramon Lopez de Mantaras (CSIC, Spain), Alberto Martelli (Torino, Italy), Ryszard Michalski (George Mason, USA), Jack Minker, (Maryland, USA), Rohit Parikh (CUNY, USA), Judea Pearl (UCLA, USA), Don Perlis (Maryland, USA), Francois G. Pinn (ORNL, USA), Henri Prade (Toulouse, France), Zbigniew W. Raś (UNC, USA), Barry Richards (Imperial College, UK), Colette Rolland (Paris I, France), Lorenza Saitta (Trento, Italy), Erik Sandewall (Linköping, Sweden), Richmond Thomason (Pittsburgh, USA), Enn Tyugu (KTH, Sweden), Ralph Wachter (ONR, USA), S.K. Michael Wong (Regina, Canada), Erling Woods (SINTEF, Norway), Maria Zemankova (NSF, USA) and Jan Zytkow (Wichita State, USA). Additionally, we acknowledge the help in reviewing the papers from: M. Beckerman, Sanjiv Bhatia, Jianhua Chen, Stephen Chenoweth, Bill Chu, Bipin Desai, Keith Downing, Doug Fisher, Melvin Fitting, Theresa Gaasterland, Atillio Giordana, Charles Glover, Diana Gordon, Jerzy Grzymala-Busse, Cezary Janikow, Kien-Chung Kuo, Rei-Chi Lee, Charles Ling, Anthony Maida, Stan Matwin,

Neil Murray, David Mutchler, Jan Plaza, Helena Rasiowa, Steven Salzberg, P.F. Spelt, David Reed, Michael Sobolewski, Stan Szpakowicz, Zbigniew Stachniak, K. Thirunarayan, Marianne Winslett, Agata Wrzos-Kamińska, Jacek Wrzos-Kamiński, Jing Xiao, Wlodek Zadrozny and Wojtek Ziarko.

The Symposium was organized by the Knowledge Systems Group of the Department of Computer Systems and Telematics, The Norwegian Institute of Technology. The Congress Department of the Institute provided the secretariat of the Symposium. The Organizing Committee consisted of Jan Komorowski, Zbigniew W. Raś and Jacek Wrzos-Kamiński.

We wish to express our thanks to François Bry, Lennart Ljung, Michael Lowry, Jack Minker, Luc De Raedt and Erik Sandewall who presented the invited addresses at the symposium. We would also like to express our appreciation to the sponsors of the symposium and to all who submitted papers for presentation and publication in the proceedings. Special thanks are due to Alfred Hofmann of Springer Verlag for his help and support.

Finally, we would like to thank Jacek Wrzos-Kamiński whose contribution to organizing this symposium was essential to its becoming a success.

March 1993

J. Komorowski, Z.W. Raś

Table of Contents

Invited Talk I

J. Minker & C. Ruiz
On Extended Disjunctive Logic Programs **1**

Logic for Artificial Intelligence I

H. Chu & D.A. Plaisted
Model Finding Stategies in Semantically Guided Instance-Based Theorem Proving **19**

D.R. Busch
An Expressive Three-Valued Logic with Two Negations **29**

J. Posegga
Compiling Proof Search in Semantic Tableaux **39**

R. Hähnle
Short CNF in Finetely-Valued Logics **49**

L. Giordano
Defining Variants of Default Logic: a Modal Approach **59**

Expert Systems

L.-Y. Shue & R. Zamani
An Admissible Heuristic Search Algorithm **69**

S.-J. Lee & C.-H. Wu
Building an Expert System Language Interpreter with the Rule Network Technique **76**

G. Valiente
Input-Driven Control of Rule-Based Expert Systems **86**

B. López & E. Plaza
Case-Based Planning for Medical Diagnosis **96**

J.P. Klut & J.H.P. Eloff
MethoDex: A Methodology for Expert Systems Development **106**

Invited Talk II

F. Bry
Towards Intelligent Databases **116**

Logic for Artificial Intelligence II

L. Padgham & B. Nebel
Combining Classification and Nonmonotonic Inheritance Reasoning: A First Step **132**

H. Rasiowa & V.W. Marek
Mechanical Proof Systems for Logic II, Consensus Programs and Their Processing (Extended Abstract) **142**

J. Chen
The Logic of Only Knowing as a Unified Framework for Nonmonotonic Reasoning **152**

P. Lambrix & R. Rönnquist
Terminological Logic Involving Time and Evolution: A Preliminiary Report **162**

Intelligent Databases

L.V. Orman
Knowledge Management by Example **172**

H.M. Dewan & S.J. Stolfo
System Reorganization and Load Balancing of Parallel Database Rule Processing **186**

T. Gaasterland & J. Lobo
Using Semantic Information for Processing Negation and Disjunction in Logic Programs **198**

P. Bosc, L. Lietard & O. Pivert
On the Interpretation of Set-Oriented Fuzzy Quantified Queries and Their Evaluation in a Database Management System **209**

Invited Talk III

M.R. Lowry
Methodologies for Knowledge-Based Software Engineering **219**

Logic for Artificial Intelligence III

N. Leone, L. Palopoli & M. Romeo
Updating Logic Programs **235**

D. Robertson, J. Agusti, J. Hesketh & J. Levy
Expressing Program Requirements using Refinement Lattices **245**

R. Chadha & D. A. Plaisted
Finding Logical Consequences Using Unskolemization **255**

A. Rajasekar
Controlled Explanation Systems 265

N.V. Murray & E. Rosenthal
Signed Formulas: A Liftable Meta-Logic for Multiple-Valued Logics 275

Approximate Reasoning

S. Tano, W. Okamoto & T. Iwatani
New Design Concepts for the FLINS-Fuzzy Lingual System: Text-Based and Fuzzy-Centered Architectures 285

A. Skowron
Boolean Reasoning for Decision Rules Generation 295

C.W.R. Chau, P. Lingras & S.K.M. Wong
Upper and Lower Entropies of Belief Functions Using Compatible Probability Functions 306

C.J. Liau & B.I-P. Lin
Reasoning About Higher Order Uncertainty in Possibilistic Logic 316

C.M. Rauszer
Approximation Methods for Knowledge Representation Systems 326

Invited Talk IV

L. Ljung
Modelling of Industrial Systems 338

Constraint Programming

J.-F. Puget
On the Satisfiability of Symmetrical Constrainted Satisfaction Problems 350

A.L. Brown Jr., S. Mantha & T. Wakayama
A Logical Reconstruction of Constraint Relaxation Hierarchies in Logic Programming 362

P. Berlandier
A Performance Evaluation of Backtrack-Bounded Search Methods for N-ary Constraint Networks 375

M.A. Meyer & J.P. Müller
Finite Domain Consistency Techniques: Their Combination and Application in Computer-Aided Process Planning 385

Learning and Adaptive Systems I

I.F. Imam & R.S. Michalski
Should Decision Trees be Learned from Examples or from Decision Rules? 395

H. Lounis
Integrating Machine-Learning Techniques in Knowledge-Based Systems Verification 405

R. Bagai, V. Shanbhogue, J.M. Żytkow & S.C. Chou
Automatic Theorem Generation in Plane Geometry 415

A. Giordana, L. Saitta & C. Baroglio
Learning Simple Recursive Theories 425

Invited Talk V

L. De Raedt & N. Lavrač
The Many Faces of Inductive Logic Programming 435

Methodologies

M. Bateman, S. Martin & A. Slade
CONSENSUS: A Method for the Development of Distributed Intelligent Systems 450

H. Gan
Script and Frame: Mixed Natural Language Understanding System with Default Theory 466

M. Franová, Y. Kodratoff & M. Gross
Contructive Matching Methodology: Formally Creative or Intelligent Inductive Theorem Proving? 476

G. Grosz & C. Rolland
Representing the Knowledge Used During the Requirement Engineering Activity with Generic Structures 486

S. Caselli, A. Natali & F. Zanichelli
Development of a Programming Environment for Intelligent Robotics 496

Knowledge Representation

A. Schaerf
On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification 508

S. Ambroszkiewicz
Mutual Knowledge 518

K. Thirunarayan
Expressive Extensions to Inheritance Networks **528**

G. Bittencourt
A Connectionist-Symbolic Cognitive Model **538**

M. Di Manzo & E. Giunchiglia
Multi-Context Systems as a Tool to Model Temporal Evolution **548**

Invited Talk VI

E. Sandewall
Systematic Assessment of Temporal Reasoning Methods for Use in Autonomous Agents **558**

Manufacturing

Ch. Klauck & J. Schwagereit
GGD: Graph Grammar Developer for Features in CAD/CAM **571**

K. Wang
A Knowledge-Based Approach to Group Analysis in Automated Manufacturing Systems **581**

B.-T.B. Chu & H. Du
CENTER: A System Architecture for Matching Design and Manufacturing **591**

M. Sobolewski
Knowledge-Based System Integration in a Concurrent Engineering Environment **601**

Learning and Adaptive Systems II

P. Charlton
A Reflective Strategic Problem Solving Model **612**

B. Wüthrich
On the Learning of Rule Uncertainties and Their Integration into Probabilistic Knowledge Bases **622**

R. Zembowicz & J.M. Żytkow
Recognition of Functional Dependencies in Data **632**

R. Słowiński
Rough Set Learning of Preferential Attitude in Multi-Criteria Decision Making **642**

Authors Index **653**

On Extended Disjunctive Logic Programs

Jack Minker[1,2] and Carolina Ruiz[1]

[1] Department of Computer Science.
[2] Institute for Advanced Computer Studies.
University of Maryland. College Park, MD 20742 U.S.A.
{minker , cruizc}@cs.umd.edu

Abstract. This paper studies, in a comprehensive manner, different aspects of extended disjunctive logic programs, that is, programs whose clauses are of the form $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$, where $l_1, ..., l_n$ are literals (i.e. atoms and classically negated atoms), and *not* is the negation-by-default operator. The explicit use of classical negation suggests the introduction of a new truth value, namely, *logical falsehood* (in contrast to *falsehood-by-default*) in the semantics. General techniques are described for extending the model, fixpoint, and proof theories of an arbitrary semantics of normal disjunctive logic programs to cover the class of extended programs. Illustrations of these techniques are given for stable models, disjunctive well-founded and stationary semantics. Also, the declarative complexity of the extended programs as well as the algorithmic complexity of the proof procedures are discussed.

1 Introduction

Logic programming, as an approach to the use of logic in knowledge representation and reasoning, has gone through different stages. First, logic programs containing only Horn clauses were considered. A *Horn clause* is a disjunction of literals in which at most one literal is positive and can be written either as: "$a \leftarrow b_1, \ldots, b_m$" or as "$\leftarrow b_1, \ldots, b_m$" where $a, b_1, \ldots, b_m$ are atoms and $m \geq 0$. The semantics of these programs is well understood (see [31, 15]) and is captured by the unique minimal Herbrand model of the program.

It is clear that since only positive atoms occur in (the head of) Horn clauses, no negative information can be inferred from these programs unless some strategy or rule for deriving negative information is adopted. Two rules for negation were initially proposed for Horn programs: The *Closed World Assumption* (CWA) [28] which states that an atom can be assumed to be *false* if it cannot be proven to be *true*; and the *Clark completion theory* [7] which assumes that the definition of each atom in a program is complete in the sense that it specifies all the circumstances under which the atom is *true* and only such circumstances, so the atom can be inferred *false* otherwise.

Having a rule for negation, it is plausible to extend Horn clauses to make use of negative information. This is the purpose of the so-called *negation-by-default* operator *not*, which may appear in the bodies of clauses. These clauses are called *normal clauses* and are of the form: "$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$" where

$a, b_1, \ldots, b_m, c_1, \ldots, c_n$ are atoms and $m, n \geq 0$. This kind of negation is limited, however, in the sense that *not* p does not refer to the presence of knowledge asserting the falsehood of the atom p but only to the lack of evidence about its truth. Indeed, some authors have translated *not* p as "p is not believed" [14], "p is not known" [10], and "there is no evidence that p is *true*" [9], in addition to the common translation "p is not provable from the program in question".

In contrast to the Horn case, there is no agreement on a unique semantics for normal programs since there can be as many different semantics as there are ways to interpret the meaning of *not*. Among the proposed semantics are the perfect model semantics [24], the stable model semantics [11], and the well-founded semantics (WFS) [32].

Another generalization of Horn clauses that allows disjunctions of atoms in the heads of clauses has been studied extensively (see [16]). These clauses are called *disjunctive clauses* and are of the following form: "$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m$" where $a_1, \ldots, a_k, b_1, \ldots, b_m$ are atoms and $k, m \geq 0$. The meaning of such a program is captured by its set of minimal Herbrand models. Several rules for negation have been introduced for disjunctive logic programs: *the Generalized Closed World Assumption* (GCWA) [20] which assumes that an atom is *false* when it does not belong to any of the minimal Herbrand models of the program, *the Extended Generalized Closed World Assumption* (EGCWA) [33] which applies exactly the same criterion of the GCWA but to conjunctions of atoms instead of only atoms (see Sect. 4) and *the Weak Generalized Closed World Assumption* (WGCWA) [27] (or equivalently, the *Disjunctive Database Rule (DDR)* [29]) which states that an atom can be assumed to be *false* when it does not appear in any disjunction derivable from the program.

Negative information can be introduced in disjunctive clauses in the same fashion as in Horn clauses. The resulting clauses are called *normal disjunctive clauses* and are of the form: "$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$" where $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are atoms and $k, m, n \geq 0$. There are also various different semantics proposed for normal disjunctive logic programs (henceforth, denoted by *ndlps*), among others, the stable disjunctive model semantics [23], the disjunctive well-founded semantics (DWFS) [2], the generalized disjunctive well-founded semantics (GWFS) [3, 4], WF^3 [5], and the stationary semantics [26].

It is worth noting that normal clauses are particular cases of disjunctive normal clauses. Therefore any semantics defined for the class of normal disjunctive logic programs is also a semantics for the class of normal logic programs.

An alternative to overcome some of the difficulties of dealing with negative information is to make explicit use of classical negation in addition to negation-by-default. In this way, the expressive power of logic programs is increased since the user is now allowed to state not only when an atom is *true* but also when it is *false* (without any ambiguity or default interpretation). Clauses obtained by explicitly using the classical negation operator ($\neg$) are called *extended disjunctive clauses* and are of the following form: "$l_1 \vee \ldots \vee l_k \leftarrow l_{k+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$," where $l_1, \ldots, l_n$ are literals (i.e. atoms and classically negated atoms), $0 \leq k \leq$

$m \leq n$. Hence, extended disjunctive clauses contain two forms of negation: classical and default.

Previous contributions in this area include the following: Pearce and Wagner [22] added explicit negative information to Prolog programs. They showed that there is no need to alter the computational structure of such programs to include classical negation since there is a way to transform extended programs to positive ones which preserves the meaning of the programs. Gelfond and Lifschitz [12] extended their stable model semantics to cover classical negation. Przymusinski [25] generalized this extended version of the stable model semantics to include disjunctive programs. Alferes and Pereira [1] provided a framework to compare the behavior of the different semantics in the presence of two kinds of negation.

The purpose of this paper is to study, in a comprehensive manner, different aspects of extended disjunctive logic programs (*edlps* for short). We describe general techniques to deal with this extended class of programs and also survey some of the results in the field.

Alternative semantics for edlps can be obtained by extending the semantics known for the class of normal disjunctive logic programs. Since there are now two different notions of falsehood in extended programs we distinguish between them by saying that, with respect to some semantics, a formula φ is *false-by-default* in an edlp P if *not* (φ) is provable from P, i.e. φ is assumed to be *false* by the particular rule for negation used by the semantics; and is *logically false* (or simply *false*) if $\neg\varphi$ is provable from P, or in other words, if $\neg\varphi$ is a logical consequence of P. We extend each semantics to include a new truth value: *logical falsehood*.

With the introduction of negated atoms in the heads of the clauses, it is possible to specify inconsistent theories, that is, to describe situations in which some atom p and its complement $\neg p$ are *true* simultaneously. Therefore, we must develop techniques to recognize when a program is inconsistent with respect to a given semantics and to deal with such an inconsistent program.

Since the techniques to be explained are general enough to be applied to any semantics of ndlps we will describe them in terms of a generic such semantics which we call *SEM*. In addition, we will illustrate the application of these techniques to the stable model semantics (covering in this way the perfect model semantics), DWFS (which covers the WFS and the minimal models semantics for disjunctive logic programs), and the stationary semantics.

The paper is organized as follows: Section 2 introduces the notation and definitions needed in the following sections. Section 3 describes a standard procedure to extend the model theoretical characterization of an arbitrary semantics of ndlps to the whole class of edlps. It includes also an illustration of this technique for the the case of the stable model semantics. Section 4 constructs a fixpoint operator to compute the extended version of a semantics *SEM* in terms of a fixpoint operator which computes the restriction of this semantics to ndlps. Illustrations are given for the DWFS and the stationary semantics. Section 5 describes a procedure to answer queries with respect to edlps and an arbitrary semantics *SEM*. This procedure uses as a subroutine, a procedure to answer

queries with respect to the restriction of *SEM* to ndlps. Section 6 studies the complexities of some fundamental problems related to edlps.

2 Syntax and Definitions

In this section we formalize the definition of extended disjunctive logic programs and introduce some of the notation needed in the following sections.

An ***extended disjunctive logic program, edlp*** , is a (possibly infinite) set of clauses of the form: $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$, where $l_1, ..., l_n$ are literals (i.e. atoms and classically negated atoms), $0 \leq k \leq m \leq n$ and *not* is the negation-by-default operator.

Example 1. The following is an extended disjunctive logic program:

$$\begin{aligned} P = \{\ & a \vee e\ ; \\ & c \leftarrow a,\ not\ b\ ; \\ & \neg b \leftarrow \neg e\ ; \\ & b \leftarrow e,\ not\ c\ ; \\ & \neg a \leftarrow not\ a \qquad \}\ . \end{aligned}$$

We assume the convention that any occurrence of $\neg\neg p$ is simplified to p.

Since a non-ground clause is equivalent to the set of all its ground instances, we consider here only ground programs (i.e. propositional programs). This is done only to simplify the notation without any loss of generality.

Given a program P, L_P denotes the ***set of predicate symbols*** that occur in P; $\mathcal{L}$ denotes the ***set of all ground literals*** that can be constructed with predicates in L_P; and $\mathcal{U}$ will denote the ***Herbrand universe*** associated with L_P.

In the context of ndlps, DHB_P (resp. CHB_P) denotes the ***disjunctive Herbrand base*** (resp. ***conjunctive Herbrand base***) of P, that is, the set of equivalence classes of disjunctions (resp. conjunctions) of atoms appearing in P modulo logical equivalence. This notion is generalized to edlps by $\mathcal{DL}_P$ (resp. $\mathcal{CL}_P$), the set of equivalence classes of disjunctions (resp. conjunctions) of literals in $\mathcal{L}$ modulo logical equivalence.[3]

As noted before, extended programs enable us not only to state when a predicate p holds but also when $\neg p$ holds. In this sense, one can regard p and $\neg p$ as different predicates which happen to be complementary (i.e. they cannot both be *true* or both be *false* at once). Using this idea, Pearce and Wagner in [22] and Gelfond and Lifschitz in [13] showed how to transform extended normal clauses into normal clauses. This is done by representing every negative literal $\neg p$ in a program by a new predicate, say p', with the restriction that p and p' cannot hold simultaneously. This restriction may be viewed as an integrity constraint.

Formally, we define the ***prime transformation*** l' of a literal l to be:

$$l' = \begin{cases} p, & \text{if } l = p \text{ for some predicate } p \\ p', & \text{if } l = \neg p \text{ for some predicate } p \end{cases}$$

[3] For simplicity, we will write d as an abbreviation for the equivalence class $[d]$.

Notice that if we apply this prime transformation to every literal occurring in an edlp P, we obtain a *normal* disjunctive logic program P'. The union of P' with the following set of integrity constraints captures the same meaning of P:

$$IC_{P'} = \{\Leftarrow p, p' : p \in L_P\}$$

These integrity constraints state that p and p' are in fact complementary predicates. We use here the symbol $\Leftarrow$ instead of $\leftarrow$ to emphasize that these integrity constraints are not clauses of the program, i.e. $IC_{P'}$ is not contained in P'.

In the same spirit, $\mathcal{L}'$ denotes the set of prime literals $\{l' : l \in \mathcal{L}\}$ and will be taken as the set of predicate symbols appearing in P', i.e. $L_{P'} =_{def} \mathcal{L}'$.

Sometimes we need to recover program P from P'. In order to do so, we define the *neg transformation* on predicates by:

$$l^{\neg} = \begin{cases} p, & \text{if } l = p \text{ for some predicate } p \\ \neg p, & \text{if } l = p' \text{ for some predicate } p \end{cases}$$

which is extended to programs in the usual way.

It is clear that for any edlp P, $(P')^{\neg} = P$ and for any ndlp Q, $(Q^{\neg})' = Q$. Also, it is worth noting that the prime transformation is not strictly needed. Instead of performing the prime transformation, we can treat $\neg a$ as if it is an atom independent of a. However, we will use this transformation in order to make explicit when an edlp P is thought of as a normal disjunctive logic program.

3 Model Theory Semantics

In this section we describe a standard procedure to extend an arbitrary model theory semantics of normal disjunctive logic programs to the whole class of extended disjunctive logic programs. Since the procedure is general enough to be applied to any semantics defined on the class of ndlps we describe it in terms of a generic such semantics which we call SEM. In the following subsection we illustrate the use of the technique when SEM is the stable model semantics.

We denote by *interpretation* any subset of the set of literals $\mathcal{L}$, and we call an interpretation *consistent* only if it does not contain any pair of complementary literals, say p and $\neg p$. The prime and neg transformations of interpretations are defined as expected: if $M \subseteq \mathcal{L}$ then $M' = \{l' : l \in M\}$ and if $N \subseteq \mathcal{L}'$ then $N^{\neg} = \{l^{\neg} : l \in N\}$.

Interpretations which agree with a given program (in the sense of the following definition) are called *models* of the program.

Definition 1. Let P be an edlp and let $M \subseteq \mathcal{L}$. Then M is a *model* of P iff M is consistent and for each program clause $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ in P, if $l_{k+1}, ..., l_m \in M$ and $l_{m+1}, ..., l_n \notin M$ then $\exists i,\ 1 \leq i \leq k$, such that $l_i \in M$.

The following Lemma establishes some relationships between the models of an edlp P and the models of P'.

Lemma 2. *Let P be an edlp, Q be a ndlp, M be an interpretation of P, and N be an interpretation of Q. The following properties hold:*

1. *M is a model of P iff M' is a model of P'.*
2. *N is a model of Q iff $N^{\neg}$ is a model of $Q^{\neg}$.*

Notice that an inconsistent edlp P (i.e. a program from which some predicate p and its complement $\neg p$ are both derivable) has no models. In accordance with classical logic, every formula is deducible from an inconsistent set of axioms, hence, we must declare the set of all literals $\mathcal{L}$ as the meaning of an inconsistent program and so, we must extend the definition of a model.

Definition 3. Let P be an edlp and let $M \subseteq \mathcal{L}$. Then M is an *extended-model* of P iff either $M = \mathcal{L}$, or M is a model of P.

Using this definition, it is easy to characterize the inconsistent edlps as those whose only extended-model is $\mathcal{L}$.

Let $\mathcal{M}_P^{SEM}$ denote the set of extended-models which characterize the semantics *SEM* of a logic program P. For instance, if *SEM* is the stable model semantics then $\mathcal{M}_P^{stable}$ is the set of all stable extended-models of P (see Sect. 3.1).

An easy way to extend *SEM* to the class of edlps is to treat each literal in an edlp program P as if it were an atom. That is, the definition of *SEM* is applied to P as if P were a normal disjunctive logic program containing some atoms which happen to begin with the character $\neg$. In this way, we obtain a set of models for P from which we must replace the ones that are inconsistent by $\mathcal{L}$.

The main result in this section states that if one uses the procedure just described to extend *SEM*, the set of extended-models which characterizes the extended semantics *SEM* of an edlp P (i.e. $\mathcal{M}_P^{SEM}$) can be obtained from the set of models characterizing the semantics *SEM* of the prime transformation of the program (i.e. $\mathcal{M}_{P'}^{SEM}$).

Theorem 4. *Let P be an edlp and let M be an interpretation of P. Then $M \in \mathcal{M}_P^{SEM}$ iff either*

1. *M is consistent and $M' \in \mathcal{M}_{P'}^{SEM}$, or*
2. *$M = \mathcal{L}$ and there is some $N \in \mathcal{M}_{P'}^{SEM}$ such that N does not satisfy $IC_{P'}$.*

$\mathcal{M}_P^{SEM}$ constructed in this way characterizes the *skeptical* version of *SEM*, that is, the version in which a formula is *true* w.r.t. P if and only if it is *true* in every $M \in \mathcal{M}_P^{SEM}$. For the *credulous* version of *SEM* in which a formula is *true* w.r.t. P if and only if it is *true* in some $M \in \mathcal{M}_P^{SEM}$, we need to reduce $\mathcal{M}_P^{SEM}$ by keeping $\mathcal{L}$ in $\mathcal{M}_P^{SEM}$ only if there is no other model in it. That is,

$$\text{reduced}(\mathcal{M}_P^{SEM}) = \begin{cases} \{\mathcal{L}\} & \text{If } \mathcal{M}_P^{SEM} = \{\mathcal{L}\}. \\ \mathcal{M}_P^{SEM} - \{\mathcal{L}\} & \text{Otherwise.} \end{cases}$$

On the other hand, the definition of $\mathcal{M}_P^{SEM}$ enables us to distinguish between a program that is inconsistent w.r.t. *SEM* , i.e. $\mathcal{M}_P^{SEM} = \{\mathcal{L}\}$ and a program which is *incoherent* w.r.t. *SEM* i.e. $\mathcal{M}_P^{SEM} = \emptyset$.

3.1 Stable Model Semantics

The stable model semantics for normal logic programs was introduced by Gelfond and Lifschitz in [11] and then generalized by them to the class of extended logic programs (see [12]). Przymusinski [25] enlarged this extended version to include disjunctive programs. Recently, Inoue et al. [14] developed a bottom-up procedure to compute stable models of edlps.

Intuitively, $M \subseteq \mathcal{L}$ is a stable extended-model of an edlp P if M is a minimal extended-model of the disjunctive program obtained from P by interpreting each negation-by-default of the form *not* l, $l \in \mathcal{L}$, according to M in the following way: *not* l is *false* if $l \in M$ and *not* l is *true* if $l \notin M$. In the first case, each clause containing *not* l in its body may be erased from P, and in the second case, any occurrence of *not* l may be deleted from clauses in P without altering the meaning of P in the context of M. If M is inconsistent, it is replaced by $\mathcal{L}$. More formally:

Definition 5. Let P be an edlp and let $M \subseteq \mathcal{L}$. The *Gelfond-Lifschitz transformation* of P w.r.t. M, P^M, is defined as:

$P^M = \{l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m \mid l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ is a clause in P and $\{l_{m+1}, ..., l_n\} \cap M = \emptyset\}$.

Definition 6. Let P be an edlp and let $M \subseteq \mathcal{L}$. Then M is a *stable extended-model* of P iff either:

1. M is consistent and M is a minimal model of P^M, or
2. $M = \mathcal{L}$ and there is some inconsistent $N \subseteq \mathcal{L}$ such that N is a minimal model of P^N.

Example 2. The stable models of the edlp P given in Example 1 are: $M_1 = \{a, c\}$ and $M_2 = \{e, b, \neg a\}$. Notice that M_1 is a minimal model of $P^{M_1} = \{a \vee e\ ;\ c \leftarrow a\ ;\ \neg b \leftarrow \neg e\}$ and M_2 is a minimal model of $P^{M_2} = \{a \vee e\ ;\ \neg b \leftarrow \neg e\ ;\ b \leftarrow e\ ;\ \neg a\}$. No other extended-model of P is stable.

The question now is how to effectively find all stable extended-models of a given edlp P. At this point we can take advantage of the fact that P' is a normal disjunctive logic program and that there are effective procedures to compute the stable models for this restricted class of programs (see [9]). We must, therefore, find a relationship between the stable extended-models of P and the stable models of P'. Such a relationship is given by the following theorem which generalizes a similar result in [13]. First we need a technical lemma stating that the Gelfond-Lifschitz transformation and the prime transformation commute.

Lemma 7. *Let P be an edlp and let $M \subseteq \mathcal{L}$. Then $(P^M)' = P'^{M'}$.*

Theorem 8. *Let P be an edlp and let $M \subseteq \mathcal{L}$. M is a stable extended-model of P iff*

1. *M is consistent and M' is a stable model of P', or*
2. *$M = \mathcal{L}$ and there is some stable model of P' which does not satisfy $IC_{P'}$.*

4 Fixpoint Semantics

In this section we construct a fixpoint operator to compute the extended version of a semantics *SEM* in terms of a fixpoint operator which computes the restriction of this semantics to ndlps.

Let T be a fixpoint operator with the property that for every ndlp Q, there exists a fixpoint ordinal α_Q such that $T_Q \uparrow \alpha_Q$ characterizes the fixpoint semantics of Q with respect to *SEM*. The desired fixpoint operator T^E to compute the semantics *SEM* of edlps can be described in two different but equivalent ways:

1. As an extension of T which works with literals instead of only positive atoms. As an illustration of this approach, we present in the following subsections the extended fixpoint operators for the disjunctive well-founded semantics and the stationary semantics.
2. As an invocation to T. More precisely, given an edlp P, the fixpoint semantics of P is obtained from the fixed-point computed by T for the ndlp P'. This can be achieved by defining $T_P^E \uparrow \alpha =_{def} (T_{P'} \uparrow \alpha)^{\neg}$ for all ordinal α. It is easy to see that if $\alpha_{P'}$ is the minimal fixpoint ordinal of $T_{P'}$ then $\alpha_{P'}$ is also the minimal fixpoint ordinal of T_P^E.
   This approach enables us to bound the complexity of computing $T_P^E \uparrow \alpha_P$ as follows: complexity($T_P^E \uparrow \alpha_P$) $\leq$ complexity(applying the prime transformation to P) + complexity($T_{P'} \uparrow \alpha_{P'}$) + complexity(applying the neg transformation to $T_{P'} \uparrow \alpha_{P'}$). Note that the complexity of transforming P to P' is linear in the size of P (for any reasonable definition of size).

The equivalence of these two approaches for the DWFS and the stationary semantics will be established in the subsections below.

In what follows, the notation T^α, where T is a fixpoint operator and α is an ordinal, stands for the composition of T with itself α times. That is, if A belongs to the domain of T, then

$$T^\alpha(A) = \begin{cases} A & \text{If } \alpha = 0. \\ T(T^{\alpha-1}(A)) & \text{If } \alpha \text{ is a succesor ordinal.} \\ \bigcup_{\beta<\alpha}(T^\beta(A)) & \text{If } \alpha \text{ is a limit ordinal.} \end{cases}$$

4.1 Disjunctive Well-Founded Semantics

The disjunctive well-founded semantics (DWFS) was defined by Baral [2] as an extension of the well-founded semantics (WFS) [32] to the class of normal disjunctive logic programs. It is also equivalent to the Minker/Rajasekar fixpoint operator on the class of disjunctive logic programs [21]. In the same spirit of WFS, DWFS is a 3-valued semantics which associates to each disjunctive normal logic program P a state–pair i.e. a tuple $S =< T_S; F_S >$ where T_S is a set of disjunctions of atoms ($T_S \subseteq DHB_P$) and F_S is a set of conjunctions of atoms ($F_S \subseteq CHB_P$) with the closure properties that if $D \in T_S, C \in F_S$ and $l \in \mathcal{L}$ then $D \vee l \in T_S$ and $C \wedge l \in F_S$. The intended meanings of T_S and F_S are as follows: T_S contains all the disjunctions that are assumed to be *true* in P

under this semantics and F_S contains all the conjunctions that are assumed to be *false-by-default* in P, i.e. the conjunctions produced by the particular default rule for negation used by this semantics.

We describe the extension of DWFS to the class of edlps by mimicking the definition of DWFS for the class of ndlps (see [16]) but working now with literals instead of only positive atoms. To do so, we need the following definitions:

Definition 9. Let P be an edlp. An *extended state-pair* S is a tuple $< T_S; F_S >$ where $T_S \subseteq \mathcal{DL}_P$, $F_S \subseteq \mathcal{CL}_P$ such that T_S is *closed under disjunctions with literals in* $\mathcal{L}$, that is, for all literals $l \in \mathcal{L}$ if $D \in T_S$ then $D \vee l \in T_S$; and F_S is *closed under conjunctions with literals in* $\mathcal{L}$, i.e. for all literals $l \in \mathcal{L}$ if $C \in F_S$ then $C \wedge l \in F_S$.

Notice that since, in the extended framework with explicit negation, we are allowed to define when an atom p is *true* as well as when it is *false* (i.e. when $\neg p$ is *true*), it is not longer adequate to say that a formula φ is *false* with respect to an extended state-pair $S =< T_S; F_S >$ if $F_S \models \varphi$. Instead, we define:

Definition 10. Let $S =< T_S; F_S >$ be an extended state-pair and let φ be a formula. The *truth value of* φ *with respect to* S is given by: φ is *true* if $T_S \models \varphi$; φ is *false* if $T_S \models \neg\varphi$; φ is *unknown* if $T_S \not\models \varphi$ and $T_S \not\models \neg\varphi$; and φ is *false-by-default* if $F_S \models \varphi$.

We say that an extended state-pair S is *inconsistent* if there exists a formula φ such that $T_S \models \varphi$ and $T_S \models \neg\varphi$.

The extended state-pair associated with an edlp P by DWFS is constructed as the smallest fixed-point of an operator $\mathcal{S}^E$ defined on the collection of extended state-pairs. Given an extended state-pair $S =< T_S; F_S >$, $\mathcal{S}^E$ augments T_S with all the disjunctions that can be deduced from P in one or more steps (i.e. in one or more applications of an immediate consequence operator defined below) under the assumption that the formulas in T_S are *true* and the ones in F_S are *false-by-default*. Similarly, $\mathcal{S}^E$ augments F_S with all the conjunctions which can be proved to be *false-by-default* in one or more steps (i.e. either the conjunction is assumed to be *false-by-default* or for each rule in P capable of deducing the conjunction, it is possible to prove that its body is *false* or *false-by-default*) supposing again that the formulas in T_S are *true* and the ones in F_S are *false-by-default*. Formally:

Definition 11. Given an edlp P,

1. *the extended state-pair assigned by* DWFS *to* P is the smallest fixed-point of the operator $\mathcal{S}^E$ defined on extended state-pairs in such a way that for any extended state-pair $S =< T_S; F_S >$:

$$\mathcal{S}^E(S) =< T_S \cup [\bigcup_{1 \le n < \omega} (\mathcal{T}_S^E)^n(\emptyset)] \; ; F_S \cup [\bigcap_{1 \le n < \omega} (\mathcal{F}_S^E)^n(\mathcal{CL}_P)] >$$

where:

$T_S^E(T) = \{D \in \mathcal{DL}_P |$ there is a ground instance of a clause $D' \leftarrow l_1, \ldots, l_m$, *not* $l_{m+1}, \ldots,$ *not* l_n in P, where $D' \in \mathcal{DL}_P$, such that for all $i, 1 \leq i \leq m$, $l_i \vee D_i \in T_S \cup T$ for some (possibly null) $D_i \in \mathcal{DL}_P$, $\{l_{m+1}, \ldots, l_n\} \subseteq F_S$ and $D' \vee D_1 \vee \ldots \vee D_m \Rightarrow$ [4] $D\}$.

$\mathcal{F}_S^E(F) = \{C \in \mathcal{CL}_P |$ for all ground instances of clauses of the form $A \vee E \leftarrow l_1, \ldots, l_m,$ *not* $l_{m+1}, \ldots,$ *not* l_n in P, where E is a (possible null) disjunction of literals and $C \Rightarrow A$, at least one of the following cases holds:

(a) $m \geq 1$ and $l_1 \wedge \ldots \wedge l_m \in F_S \cup F$

(b) $n \geq m + 1$ and $l_{m+1} \vee \ldots \vee l_n \in T_S\}$.

2. *the extended state-pair characterizing the semantics* DWFS *of* P is defined as the extended state-pair S assigned by DWFS to P if S is consistent and as $< \mathcal{DL}_P; \mathcal{CL}_P >$ otherwise. In the latter case, the program P is called *inconsistent* w.r.t. DWFS.

In order to simplify the notation in the following examples, we use the following convention: If $T \subseteq \mathcal{DL}_P$ then *disj-closure*$_{\mathcal{L}}(T)$ denotes the smallest subset of $\mathcal{DL}_P$ containing T that is closed under disjunctions with literals in $\mathcal{L}$. If $F \subseteq \mathcal{CL}_P$ then *conj-closure*$_{\mathcal{L}}(F)$ denotes the smallest subset of $\mathcal{CL}_P$ containing F that is closed under conjunctions with literals in $\mathcal{L}$.

Example 9. The extended state-pair characterizing the DWFS of the edlp given in Example 1 is:

$$S = < \text{disj-closure}_{\mathcal{L}}(\{a \vee e\}) ; \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e\}) >,$$

which is obtained from:

$$\begin{aligned}
&T^E_{<\emptyset;\emptyset>}(\emptyset) &&= \text{disj-closure}_{\mathcal{L}}(\{a \vee e\}) = (T^E_{<\emptyset;\emptyset>})^2(\emptyset)\\
&\mathcal{F}^E_{<\emptyset;\emptyset>}(\mathcal{CL}_P) &&= \text{conj-closure}_{\mathcal{L}}(\{b, \neg b, c, \neg c, \neg e\})\\
&(\mathcal{F}^E_{<\emptyset;\emptyset>})^2(\mathcal{CL}_P) &&= \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e\}) = (\mathcal{F}^E_{<\emptyset;\emptyset>})^3(\mathcal{CL}_P). \text{ Then,}\\
&\mathcal{S}^E(<\emptyset;\emptyset>) &&= < \text{disj-closure}_{\mathcal{L}}(\{a \vee e\}) ; \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e\}) >\\
& &&= (\mathcal{S}^E)^2(<\emptyset;\emptyset>).
\end{aligned}$$

Theorem 12. *Let P be an edlp consistent w.r.t. DWFS and let $S =< T_S; F_S >$ be an extended state-pair. Then S characterizes the semantics DWFS of P iff $S' =< T'_S; F'_S >$ characterizes the semantics DWFS of P' and T'_S satisfies $IC_{P'}$.*

A similar approach can be followed to extend the generalized disjunctive well-founded semantics (GDWFS) and WF^3 to the class of edlps.

[4] "$\Rightarrow$" denotes logical implication.

4.2 Stationary Semantics

The stationary semantics introduced by Przymusinski [26] also associates to each ndlp a state-pair. The construction of this state–pair (see [16]) relies on the notions of Extended Generalized Closed World Assumption (EGCWA) [33] and stationary transformation, which we generalize to the extended case as follows:

Definition 13. Let P be an edlp and let S be an extended state–pair. The *stationary transformation of P with respect to S*, denoted by *Sta(P,S)*, is the edlp free of negation-by-default[5] obtained from P by:

1. removing each clause in P whose body is *false*[6] or *false-by-default* with respect to S.
2. removing the bodies of the remaining clauses in P.

An easy way to enlarge the EGCWA to the class of edlps free of negation-by-default is by using the prime transformation described in Sect. 2.

Definition 14. Let P be an edlp free of negation-by-default. We define the $EGCWA^E(P)$ as $(EGCWA(P'))^{\neg}$, i.e.

$EGWA^E(P) = \{not\ l_1 \vee \ldots \vee not\ l_n \mid l_1, \ldots l_n \in \mathcal{L}$ and $not\ l'_1 \vee \ldots \vee not l'_n$ is *true* in every minimal model of $P'\}$.

The extended state–pair which characterizes the stationary semantics of an edlp P is the smallest fixed-point of an operator $\mathcal{S}^{SE}$ defined on the set of extended state-pairs in such a way that for any given extended state–pair $S =< T_S; F_S >$, $\mathcal{S}^{SE}$ adds to T_S the set T_S^{SE} of all the disjunctions that are logical consequences of the union of P with T_S and the negation of the conjunctions in F_S. On the other hand, $\mathcal{S}^{SE}$ adds to F_S all the conjunctions that can be assumed to be *false-by-default* under the $EGCWA^E$ from the stationary transformation of P with respect to S and T_S^{SE}.

In what follows, $not\ l_1 \vee \ldots \vee not\ l_n$ is abbreviated as $not(l_1 \wedge \ldots \wedge l_n)$; if C is a set of conjunctions of literals then $not(C)$ will denote the set $\{not(c) | c \in C\}$; and $not(not\ p) = p$.

Definition 15. Given an edlp P,

1. *the extended state–pair assigned by the stationary semantics to P* is the smallest fixed-point of the operator $\mathcal{S}^{SE}$ defined on extended state-pairs in such a way that for any extended state–pair $S =< T_S; F_S >$:

$$\mathcal{S}^{SE}(S) =< T_S \cup T_S^{SE}; F_S \cup F_S^{SE} >$$

where:
$T_S^{SE} = \{D \in \mathcal{DL}_P | P \cup T_S \cup not(F_S) \models D\}$ and
$F_S^{SE} = not(EGCWA(Sta(P, < T_S^{SE} \cup T_S; F_S >) \cup T_S^{SE} \cup T_S)$.

---

[5] By this, we mean that the operator *not* does not appear in the program.

[6] The truth value of a formula with respect to an extended state–pair is given in Def. 10.

2. *the extended state-pair characterizing the stationary semantics of* P is defined as the extended state–pair S assigned by the stationary semantics to P if S is consistent and as $< \mathcal{DL}_P; \mathcal{CL}_P >$ otherwise. In the latter case, the program P is called *inconsistent* w.r.t. the stationary semantics.

Example 4. The extended state–pair which characterizes the stationary semantics of the edlp P given in Example 1 is:

$$S =< \text{disj-closure}_{\mathcal{L}}(\{a \vee e\}) ; \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e, a \wedge e\}) >,$$

which is obtained from:

$$\begin{aligned}
\mathcal{T}^{SE}_{<\emptyset;\emptyset>} &= \text{disj-closure}_{\mathcal{L}}(\{a \vee e\}). \text{ Let us call this set D.}\\
\mathcal{F}^{SE}_{<\emptyset;\emptyset>} &= not(EGCWA^E(\{a \vee e, c, \neg b, b, \neg a\} \cup \{a \vee e\}))\\
&= \text{conj-closure}_{\mathcal{L}}(\{\neg c, \neg e, a \wedge e\}). \text{ Let us call this set } \mathcal{C}. \text{ Then,}\\
\mathcal{S}^{SE}(<\emptyset;\emptyset>) &= <\mathcal{D};\mathcal{C}> \textit{ and}\\
\mathcal{T}^{SE}_{<\mathcal{D};\mathcal{C}>} &= \text{disj-closure}_{\mathcal{L}}(\{a \vee e\})\\
\mathcal{F}^{SE}_{<\mathcal{D};\mathcal{C}>} &= not(EGCWA^E(\{a \vee e, c, b, \neg a\} \cup \{a \vee e\}))\\
&= \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e, a \wedge e\}). \text{ Therefore,}\\
(\mathcal{S}^{SE})^2(<\emptyset;\emptyset>) &= \mathcal{S}^{SE}(<\mathcal{D};\mathcal{C}>)\\
&= <\text{disj-closure}_{\mathcal{L}}(\{a \vee e\}); \text{conj-closure}_{\mathcal{L}}(\{\neg b, \neg c, \neg e, a \wedge e\}) >\\
&= (\mathcal{S}^{SE})^3(<\emptyset;\emptyset>).
\end{aligned}$$

Theorem 16. *Let P be an edlp consistent w.r.t. the stationary semantics and let $S =< T_S; F_S >$ be an extended state–pair. Then S characterizes the stationary semantics of P iff $S' =< T'_S; F'_S >$ characterizes the stationary semantics of P' and T'_S satisfies $IC_{P'}$.*

5 Proof Theory

In this section we describe a procedure to answer queries w.r.t. edlps and an arbitrary semantics *SEM*. This procedure uses as a subroutine, a procedure to answer queries w.r.t. ndlps and *SEM*. Notice that to obtain an effective proof procedure we must restrict ourselves to work with programs containing only a finite number of clauses. However, this is not a strong restriction since the proof procedure described below is capable of dealing with non-ground programs.

We will allow queries q of the following sort:

- $q(\mathbf{X}) = l(\mathbf{X})$, where l is a literal.
- $q(\mathbf{X}) = not\ l(\mathbf{X})$, where l is a literal.
- $q(\mathbf{X}) = q_1(\mathbf{X}) \wedge q_2(\mathbf{X})$, where q_1 and q_2 are queries.
- $q(\mathbf{X}) = q_1(\mathbf{X}) \vee q_2(\mathbf{X})$, where q_1 and q_2 are queries.

where $\mathbf{X} = X_1, \ldots, X_n$ lists all the free variables in q which are interpreted as being existentially quantified.

We define the correct answer to a query as *true*, *false*, or *unknown* according to the following definition:

Definition 17. Given a query q we define the *correct answer* to the query with respect to a consistent edlp P and semantics SEM to be the truth value determined as follows:

1. If q is ground then:

$$answer(P,q) = \begin{cases} true, & \text{for all } M \in \mathcal{M}_P^{SEM},\ M \models q \\ false, & \text{for all } M \in \mathcal{M}_P^{SEM},\ M \models \neg q \\ unknown, & \text{otherwise.} \end{cases}$$

2. If $q = q(\mathbf{X})$, where $\mathbf{X} = X_1, \ldots, X_n$ lists all the free variables in q, then:

$$answer(P,q(\mathbf{X})) = \begin{cases} true, & \text{if there exists } \mathbf{a} \in \mathcal{U}^n \text{ s.t. } answer(P,q(\mathbf{a})) = true \\ false, & \text{if for all } \mathbf{a} \in \mathcal{U}^n \ answer(P,q(\mathbf{a})) = false \\ unknown, & \text{otherwise.} \end{cases}$$

In the unlikely event that *false-by-default* is considered as an appropriate answer to be given to an user, it can be defined as $answer(P,q) = false\text{-}by\text{-}default$ if $answer(P, not(q)) = true$, where $not(q)$ stands for the logically equivalent query to *not* q in which the operator *not* appears only in front of atomic formulas.

And finally, we define what we mean by $M \models q$ as follows:

Definition 18. Given a model M and a query q, we define $M \models q$ as:

1. If q is ground then:
   - If $q = l$, where l is a literal: $M \models l$ iff $l \in M$.
   - If $q = not\ l$, where l is a literal: $M \models not\ l$ iff $l \notin M$.
   - If $q = q_1 \wedge q2$, where q_1 and q_2 are queries: $M \models q_1 \wedge q_2$ iff $M \models q_1$ and $M \models q_2$.
   - If $q = q_1 \vee q2$, where q_1 and q_2 are queries: $M \models q_1 \vee q_2$ iff $M \models q_1$ or $M \models q_2$.
2. Otherwise if $q = q(\mathbf{X})$, where $\mathbf{X} = X_1, \ldots, X_n$ lists all the free variables in q, then:
   - $M \models q(\mathbf{X})$ iff $M \models q(\mathbf{a})$ for every $\mathbf{a} \in \mathcal{U}^n$.

Our main purpose now is to define a sound and complete proof procedure EPP_{SEM}, to answer queries w.r.t. edlps and SEM. To do so, we assume we have a proof procedure to answer queries w.r.t. ndlps and SEM, which we call PP_{SEM}. For clarity, we denote by $PP_{SEM}(Q,q)$, the answer of the procedure PP_{SEM} to query q w.r.t. a ndlp Q and by $EPP_{SEM}(P,q)$, the answer of the procedure EPP_{SEM} to query q w.r.t. an edlp P. Remember that an edlp (in contrast with a ndlp) can be inconsistent. In that case the answer to any query should be *true*. To take that into consideration we introduce the following definition.

Definition 19. Given an edlp P, we define the ndlp P^{IC} as:

$$P^{IC} = P' \cup \{\bot \leftarrow p, p' \mid p \text{ is a predicate symbol in } L_P\},$$

where $\bot$ is a new predicate symbol not in L_P.

Given an edlp P, we can determine whether or not this program is inconsistent w.r.t. *SEM* using a sound and complete proof procedure PP_{SEM} and the following characterization:

P is inconsistent w.r.t. *SEM* iff $PP_{SEM}(P^{IC}, \bot) = \mathit{true}$.

In this way, determining whether the program is inconsistent can be done as preprocessing, previous to any attempt to answer queries.

Now we are ready to describe the extended proof procedure.

Definition 20. Let PP_{SEM} be a proof procedure to answer queries w.r.t. ndlps and *SEM*. We define an extension of this procedure, called EPP_{SEM}, which answers queries w.r.t. edlps and *SEM* as follows:

Given an edlp P and a query q do:

- If P is inconsistent w.r.t. *SEM* then $EPP_{SEM}(P, q) = \mathit{true}$.
- Otherwise:
  1. Simultaneously apply PP_{SEM} to the query $((q)' \vee \bot)$ and the query $((\neg q)' \vee \bot)$[7] w.r.t. program P^{IC}.
  2. If $PP_{SEM}(P^{IC}, (q)' \vee \bot) = \mathit{true}$ then $EPP_{SEM}(P, q) = \mathit{true}$.
  3. If $PP_{SEM}(P^{IC}, (\neg q)' \vee \bot) = \mathit{true}$ then $EPP_{SEM}(P, q) = \mathit{false}$.
  4. Otherwise $EPP_{SEM}(P, q) = \mathit{unknown}$.

In can be shown that EPP_{SEM} is well defined and gives the correct answer (in the sense of Def. 17). The following example illustrates the need of "$\bot$" in the previous definition.

Example 5. Consider the edlp $P = \{a \vee b \ ; \ a \vee \neg b \ ; \ \neg a \vee b\}$. Hence, $P' = \{a \vee b \ ; \ a \vee b' \ ; \ a' \vee b\}$ and $P^{IC} = \{a \vee b \ ; \ a \vee b' \ ; \ a' \vee b; \bot \leftarrow a, a' \ ; \bot \leftarrow b, b'\}$. The set of stable models of P^{IC} is $\mathcal{M}_{P^{IC}}^{stable} = \{\{a, b\}, \{a, a', \bot\}, \{b, b', \bot\}\}$ and $\mathcal{M}_{P}^{stable} = \{\{a, b\}, \mathcal{L}\}$. Notice that a is *true* w.r.t. P and the stable model semantics but a does not hold in every minimal model in $\mathcal{M}_{P^{IC}}^{stable}$ since it fails to hold in $\{b, b', \bot\}$. It is clear, however, that a holds in every consistent (w.r.t. $IC_{P'}$) model in $\mathcal{M}_{P^{IC}}^{stable}$. Therefore $(a \vee \bot)$ is *true* w.r.t. to P^{IC} and the stable model semantics, hence by definition 20, $EPP_{stable}(P, a) = PP_{stable}(P^{IC}, (a \vee \bot) = \mathit{true}$ as desired.

The complexity of EPP_{SEM} can be expressed in terms of the complexity of PP_{SEM} as follows: complexity(EPP_{SEM}) = 2 complexity(PP_{SEM}) + O(n), where n is the size of q and O(n) corresponds to the complexity of translating q to q' and to $(\neg q)'$.

Theorem 21. *Let PP_{SEM} and EPP_{SEM} be as in Def. 20. The following statements hold:*

[7] By $\neg q$ we mean the logically equivalent query to $\neg q$ in which $\neg$ appears only in front of atomic formulas. It is obtained by applying De Morgan's laws as needed.

1. *If PP_{SEM} is a sound proof procedure w.r.t. normal disjunctive logic programs so is EPP_{SEM} w.r.t. extended disjunctive logic programs.*
2. *If PP_{SEM} is a complete proof procedure w.r.t. normal disjunctive logic programs so is EPP_{SEM} w.r.t. extended disjunctive logic programs.*

This general extension may be applied, for instance, to the proof procedure developed by Fernández and Lobo [8] to answer queries w.r.t. ndlps and the stable model semantics.

6 Complexity

The complexity of different problems related to normal logic programs has been studied extensively (see e.g. [6],[30]). In this section we show how to express the complexities of three fundamental problems for edlps in terms of the corresponding complexities for ndlps. Also, some complexity results for the stable model semantics are surveyed here and extended to cover logic programs with classical negation.

In what follows, P denotes a finite edlp, and n denotes the size of the program (which can be measured, for instance, as the number of clauses in the program or as the number of literals appearing in it). Also, a *normal literal* denotes either a literal $l \in \mathcal{L}$ or the negation-by-default, *not* l, of such a literal.

Determining if a ground normal literal is true in some model in $\mathcal{M}_P^{SEM}$. Let us denote this problem by *credulous-truth(SEM, edlps)*.

As shown in Sect. 5, a ground normal literal g is *true* in some $M \in \mathcal{M}_P^{SEM}$ if and only if $(g' \vee \bot)$ is *true* in some $N \in \mathcal{M}_{PIC}^{SEM}$. Therefore, to determine credulous-truth of g in $\mathcal{M}_P^{SEM}$, it is enough to transform P to P^{IC} and g to g' (which clearly can be done in linear time on the size of the program), and then checking credulous-truth of g' and (if necessary) credulous-truth of $\bot$ in $\mathcal{M}_{PIC}^{SEM}$. Then,

$$\begin{aligned} &\text{complexity(credulous-truth}(SEM, \text{ ndlps})) \\ \leq\ &\text{complexity(credulous-truth}(SEM, \text{ edlps})) \\ \leq\ &2 \text{ complexity(credulous-truth}(SEM, \text{ ndlps}) + O(n). \end{aligned}$$

The first inequality holds since any ndlp is also an edlp. As an illustration, consider the stable model semantics. Marek and Truszczyński showed in [19] that for propositional normal logic programs (prop-nlps), credulous-truth(stable, prop-nlps) is NP-complete. Therefore, for propositional extended normal logic programs (prop-enlps) credulous-truth(stable, prop-enlps) is also NP-complete.

Determining if a ground normal literal is true in every model in $\mathcal{M}_P^{SEM}$. Let us denote this problem by *skeptical-truth(SEM, edlps)*.

Notice that a ground normal literal g is *true* in every model in $\mathcal{M}_P^{SEM}$ if and only if *not* g does not hold in any model in $\mathcal{M}_P^{SEM}$. Hence,

complexity(skeptical-truth(SEM, edlps))
= complexity(complement(credulous-truth(SEM, edlps)).

Then, for the stable model semantics and propositional normal logic programs: complexity(skeptical-truth(SEM, prop-enlps)) = co-NP-complete. which extends the result in [19] to extended logic programs.

Determining if a program has a model w.r.t. SEM. Notice that by Theorem 4, an edlp P has a model w.r.t. SEM if and only if P' does. Therefore:

complexity(checking existence of a model of P w.r.t. SEM)
= complexity(checking existence of a model of P' w.r.t. SEM) + O(n).

O(n) in the previous equation comes from translating P to P'. For propositional normal logic programs, Marek and Truszczyński [18] proved that determining the existence of stable models is NP-complete, hence, for the class of propositional extended normal logic programs this problem is also NP-complete. Marek, Nerode and Remmel [17] showed that the set of Gödel numbers of finite predicate normal logic programs is a Σ_1^1-complete set. It is easy to see that their result also holds for the set of Gödel numbers of finite predicate *extended* normal logic programs since the prime transformation is recursive and indeed can be performed in polynomial time.

7 Conclusions

Two criticisms have been made about the use of classical negation in logic programming. The first one states that there may be cases in which it is not feasible to include every piece of negative information in the domain of the problem. In this regard we note that Gelfond and Lifschitz [12] have pointed out that when the positive information about some predicate p is complete in a program (that is, all conditions under which p is *true* are given in the program), it is enough to define $\neg p$ as "$\neg p \leftarrow$ *not* p". This remark also justifies, to some extent, keeping negation-by-default in extended logic programs.

The second criticism addresses the potential inconsistency of extended logic programs. In this respect we have described here concrete mechanisms to deal with inconsistent logic programs.

These difficulties seem to be a small price to pay for the increased expressive power gained by the explicit use of classical negation in logic programs.

8 Acknowledgements

Support for this paper was provided by the Air Force Office of Scientific Research under grant number 91-0350 and the National Science Foundation under grant number IRI-8916059.

References

1. J.J. Alferes and L.M. Pereira. On logic program semantics with two kinds of negation. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 574–588, Washington, D.C. USA, Nov 1992. The MIT Press.
2. C. Baral. *Issues in Knowledge Representation: Semantics and Knowledge Combination*. PhD thesis, University of Maryland, College Park, MD. 20742 USA, 1991.
3. C. Baral, J. Lobo, and J. Minker. Generalized disjunctive well-founded semantics: Declarative semantics. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems*, pages 465–473, Knoxville TN, USA, 1990.
4. C. Baral, J. Lobo, and J. Minker. Generalized disjunctive well-founded semantics: Procedural semantics. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems*, pages 456–464, Knoxville TN, USA, 1990.
5. C. Baral, J. Lobo, and J. Minker. WF^3: A semantics for negation in normal disjunctive logic programs. In *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems*, pages 459–468, Charlotte NC, USA, 1991.
6. M. Cadoni and M. Schaerf. A survey on complexity results for non-monotonic logics. Preprint, Università di Roma "La Sapienza", via Salaria 113, 00198 Roma, Italy, 1992.
7. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, New York, USA, 1978.
8. J.A. Fernández and J. Lobo. A proof procedure for stable theories. Technical Report CS-TR-3034, UMIACS-TR-93-14, University of Maryland, College Park, MD 20742 USA, 1993.
9. J.A. Fernández, J. Lobo, J. Minker, and V.S. Subrahmanian. Disjunctive lp + integrity constrains = stable model semantics. *Annals of Mathematics and Artificial Inteligence*, 8(3-4), 1993.
10. M. Gelfond. On stratified autoepistemic theories. In *Proceedings of AAAI-87*, pages 207–211, 1987.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Procedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle, WA. USA, Aug. 1988. The MIT Press.
12. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D.H.D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 579–597, Jerusalem, Israel, June 1990. The MIT Press.
13. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
14. K. Inoue, M. Koshimura, and R. Hasegawa. Embedding negation as failure into a model generation theorem prover. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, pages 400–415, Saratoga Springs NY, USA, June 1992. Springer-Verlag.
15. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second extended edition, 1987.
16. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. The MIT Press, 1992.

17. V. W. Marek, A. Nerode, and J.B. Remmel. The stable models of a predicate logic program. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 446–460, Washington, USA, Nov 1992. The MIT Press.
18. V. W. Marek and Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.
19. V. W. Marek and Truszczyński. Computing intersection of autoepistemic expansions. In *Proceedings of the Fifth International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 37–50. The MIT Press, 1991.
20. J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of the Sixth Conference on Automated Deduction*, pages 292–308, 1982.
21. J. Minker and A. Rajasekar. A fixpoint semantics for disjunctive logic programs. *Journal of Logic Programming*, 9(1):45–74, July 1990.
22. P. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Proceedings of the International Workshop on Extensions of Logic Programming*, pages 311–326, Tübingen, FRG, Dec. 1989. Lecture Notes in Artificial Intelligence, Springer -Verlag.
23. T. C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9:401–424, 1991.
24. T.C. Przymusinski. Perfect model semantics. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1081–1096, Seattle, WA. USA, Aug. 1988. The MIT Press.
25. T.C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In D.H.D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 459–477, Jerusalem, Israel, June 1990. The MIT Press.
26. T.C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In S. Debray and M. Hermenegildo, editors, *Proceedings of the North American Conference on Logic Programming*, pages 42–59, Austin, TX. USA, Oct. 1990. The MIT Press.
27. A. Rajasekar, J. Lobo, and J. Minker. Weak generalized closed world assumption. *Automated Reasoning*, 5:293–307, 1989.
28. R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum, New York, 1978.
29. K.A. Ross and R.W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4(2):397–424, Dec. 1988.
30. J.S. Schlipf. A survey of complexity and undecidability results in logic programming. In H. Blair, V.W. Marek, A. Nerode, and J. Remmel, editors, *Informal Proceedings of the Worshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming.*, pages 143–164, Washington, D.C. USA, Nov. 1992.
31. M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.
32. A. van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems.*, pages 221–230, 1988.
33. A. Yahya and L.J. Henschen. Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1(2):141–160, 1985.

Model Finding Strategies in Semantically Guided Instance-based Theorem Proving*

Heng Chu and David A. Plaisted

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175, USA
Email: {chu|plaisted}@cs.unc.edu
Phone numbers: 919-962-{1733|1751}

Abstract. Semantic hyper-linking has recently been proposed [1] to use semantics in an instance-based theorem prover. The basic procedure is to generate ground instances of the input clauses until the ground clause set is unsatisfiable. Models for the satisfiable ground set are constructed periodically to guide generation of the new ground instances to change the models, until no model can be constructed. In this paper we discuss some model finding strategies that can generate useful ground instances, without using semantics, to change the ground models. We show that such strategies are helpful and will not increase the search space of the semantic hyper-linking. In addition, using semantics is often expensive. Since semantics is not used in those model finding strategies, they help to find the proofs earlier and faster.

1 Introduction

Instance-based theorem proving is a direct application of Herbrand's theorem [4]. The basic idea is to generate ground instances of the input clauses and check the satisfiability of the ground clause set using a propositional calculus (PC)prover. Early results [3] were not impressive. Lee and Plaisted [6] recently developed a fast instance-based theorem proving procedure, ***hyper-linking***, to generate instances (not necessarily ground) of the input clauses using unification. For a clause C, an instance $C\theta$ is generated if for each literal L_i of C, there is one literal R_i in some other clause such that $L_i\theta = \sim R_i\theta$, where θ is the most general such substitution. We say L_i and R_i are ***hyper-linked***. Those instances are then *grounded* (replacing all variables by a constant) and submitted to a PC prover to check satisfiability. Since, unlike resolution, literals from different clauses are not combined, hyper-linking performs well on non-Horn problems. Results [5, 6] show that hyper-linking is a useful theorem proving technique.

* This research was partially supported by the National Science Foundation under grant CCR-9108904

Chu and Plaisted [1] developed *semantic hyper-linking* to use semantics with hyper-linking. Fast proofs on hard theorems like IMV and AM8, which ordinary hyper-linking cannot prove due to large search space, show that semantic hyper-linking provides a practical method to use semantics in theorem proving.

Semantic hyper-linking is different from hyper-linking: In semantic hyper-linking, models found by the PC prover for the ground clause sets are used to guide the next semantic hyper-linking to generate more ground instances of the input clauses until the ground clause set is unsatisfiable. Since semantic hyper-linking uses user-provided semantics and often semantic checks are time consuming, we want to check that, without using semantics, the models found for the ground clause set cause no contradictions with the (non-ground) input clauses. However, this problem is undecidable in general.

In this paper, we discuss the model finding strategies, called *incremental model finding*, used in semantic hyper-linking to search for useful models of the ground clause set. In the next two sections, we will describe briefly the PC prover and semantic hyper-linking. Then we will discuss the incremental model finding strategies in detail. In the last section to conclude the paper, we discuss the results using the model finding strategies to prove two hard theorems.

1.1 Propositional Calculus (PC) Prover

The PC prover [5] used in hyper-linking is a modified Davis-Putnam procedure [2]. The basic step is to apply (recursively) case analysis and simplification on the ground clause set. Simplification rules are described below, where S is a conjunction of clauses and A is a disjunction of literals.

$$\frac{S \wedge (A \vee TRUE)}{S} \ (Unit\ Deletion)$$

$$\frac{S \wedge (A \vee FALSE)}{S \wedge A} \ (Unit\ Simplification)$$

$$\frac{S \wedge TRUE}{S}$$

$$\frac{S \wedge FALSE}{FALSE}$$

Backtracking, either to another case or previous case analysis, occurs when simplification gives a FALSE (contradiction). When S is TRUE, a *model* is constructed to be the set of literals chosen for case analysis.

The PC prover procedure is a preorder traversal of a *case tree* [5]. For a ground clause set S, let A be the set of distinct atoms occurring in S. A case tree is a binary tree T in which each edge is an atom from A or its negation, each non-terminal node contains two subtrees linked by two edges attached with L and $\sim L$ respectively and $L \in A$, and any path from the root contains no complementary pairs. Each node of a case tree can be seen as containing two cases: TRUE and FALSE of a literal L. The case tree is not unique due to the different orderings to choose literals for case analysis.

1.2 Semantic Hyper-Linking

In this section we briefly describe semantic hyper-linking. More detailed discussions are in [1].

Semantic hyper-linking is a refutation procedure employing semantics in an instance-based theorem prover. A semantic *structure* $\mathcal{S}_\mathcal{M}$ is given as input with the axioms and the negation of the theorem. The structure contains the domain and interpretation for the theorem. In general, a structure S can be seen as a (probably infinite) set of all ground literals *true* in S. If the input clause set is unsatisfiable, we can generate some ground instances that are false in $\mathcal{S}_\mathcal{M}$. We then find a model M_1 for those ground instances. There must be some literals true in M_1 but false in $\mathcal{S}_\mathcal{M}$; we call these literals the *eligible literals*. $\mathcal{S}_\mathcal{M}$ and M_1 constitute a new structure $\mathcal{S}_1$ which is the same as $\mathcal{S}_\mathcal{M}$ except for the eligible literals. Eligible literals are then used as guidance to generate new ground instances that are false in $\mathcal{S}_1$. A new model M_2 is found for the new set of ground clauses. Literals true in M_2 but false in $\mathcal{S}_1$ are the new eligible literals. M_2 and $\mathcal{S}_\mathcal{M}$ constitute another new structure $\mathcal{S}_2$. We exhaustively search for models M_3, M_4, ... in this fashion until the ground clause set is unsatisfiable and a proof is found. Each M_i and $\mathcal{S}_\mathcal{M}$ constitute a new structure $\mathcal{S}_i$; semantic hyper-linking generates ground instances false in $\mathcal{S}_i$. The process of generating ground instances from $\mathcal{S}_i$ is called a *round*. The input semantic structure $\mathcal{S}_\mathcal{M}$ is used to filter out unwanted instances and prune the search space dramatically.

When a model M_i is found for the ground clause set, sometimes ground instances can be derived from the input (non-ground) clauses to contradict a model M_i and obtain a new model M_{i+1}. Such changes to M_i are syntactic and generate ground instances without using semantics; they are thus faster since using semantics usually is expensive. In the next section we will discuss some strategies used to contradict a model for the ground clauses by generating ground instances or ground logical consequences of the input clauses.

2 Model Finding

In searching for a model for the ground clause set, we use a method called *incremental model finding*. The old model is kept and we search for a new model based on the old one. The idea is: We continue the PC prover from the node in the case tree where the old model is found last time. New models are constructed *incrementally* in such a way that no old model ever becomes a subset of a new model, and the same path from the root of the case tree is never visited twice. The model finding is incremental because new ground clauses might gradually "expand" the case tree (which is usually infinite).

During the search for a model of the ground clause set, three different methods are applied to detect and discard models causing contradictions: *model filtering*, *model literal replacement* and *UR resolution*. Those methods check if a model contradicts the input clauses. In general, however, such task is undecidable.

2.1 Incremental Model Finding

The incremental model finding method is a modified PC prover. One major difference is that it starts with an old model found earlier and builds a new model based on the old one; the PC prover in Lee's prover stops as soon as the clause set is found to be satisfiable. Another difference is that we maintain only *one* case tree (possibly infinite) throughout the whole search of proof. In each incremental model finding, backtracking takes place on failure nodes (which contain FALSE); subtrees of failure nodes are discarded; paths containing failure nodes are never visited again; and the case tree is expanded when necessary.

Like the PC prover, model finding can be thought of as a preorder traversal of the case tree starting from the root. The two sub-cases of each node represent the case analysis on one literal: one is true in the user-provided semantics, the other is false. To search for a new model when new ground clauses are generated, we continue from the node where the previous model is found. Additionally, in case analysis the true case is always chosen first. This is a heuristic method to reduce the number of eligible literals in the model.

For an example about how incremental model finding works, consider the case tree [1] in Fig. 1 for a set S of ground clauses. $\times$ indicates a failure node; the model $M_1 = \{ s(b), \sim s(i(b)), \sim s(e), p(e, i(b), i(b)) \}$ is found at node N_1. With M_1, suppose semantic hyper-linking generates a new clause $C = \{s(e), \sim s(b), \sim s(b), \sim p(b, i(b), e)\}$. If we continue the model finding at node N_1 after C is added which contradicts M_1, we can get the case tree in Fig. 2. Note that at node N_2, a new model $M_2 = M_1 \cup \{ \sim p(b, i(b), e) \}$ is found. The case tree is "expanded" in Fig. 2 to include case analysis on the new literal $p(b, i(b), e)$.

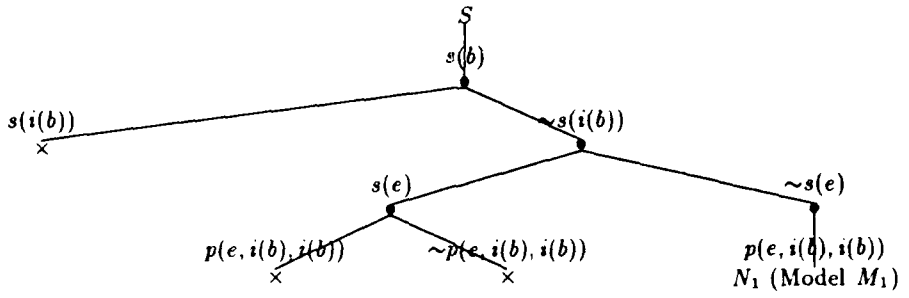


Fig. 1. The case tree when model M_1 is found at node N_1

2.2 Model Filtering

After a model is found for the ground clause set, it might directly contradict the input clauses, namely, there might be ground instances (not yet generated) of the input clauses which falsify the model. We use *model filtering* to filter out such models and make sure that the model we found satisfies the input clauses

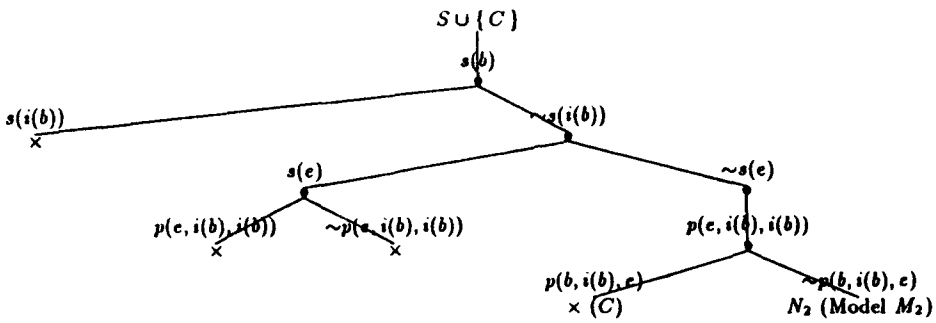


Fig. 2. The case tree is expanded and model M_2 is found at node N_2

also. The filtering idea is simple: We use the input clauses to hyper-link with the literals in the model. If one input clause C is fully hyper-linked with the model literals to get $C\theta$ where θ is a ground substitution (because all literals in the model are ground), the model contradicts $C\theta$. We then add $C\theta$, which is a ground instance of an input clause C, in the ground clause set to avoid generating the models causing the same contradiction.

For example, suppose we obtain a model $\{p(a), q(b), \sim r(a,b), \ldots\}$. Suppose there exists an input clause $\{\sim p(X), \sim q(Y), r(X,Y)\}$. The model contradicts this clause and thus should not be kept; the model finding should backtrack to find another model. The ground clause $\{\sim p(a), \sim q(b), r(a,b)\}$ is added to the ground clause set to avoid generating any other model containing literals $p(a), q(b)$, and $\sim r(a,b)$.

If more than one input clause is contradicted by the model, we only add the smallest ground instance of those to the ground clause set. In general it is not necessary to add all contradicted instances because the model will be changed and probably remove all other contradictions. Another reason is that our model finding is fast and contradicted ground instances will be added eventually if they still cause contradictions in the new models.

In fact those contradicted ground instances can be found in later rounds of semantic hyper-linking if model filtering is not used. However, model filtering detects the contradictions as early as possible to save some rounds of semantic hyper-linking and find the proofs sooner.

Manthey and Bry [7] used the same technique to check a model in their model-generation theorem prover SATCHMO. They used a different way to generate ground literals, which constitute a model for the input clauses. In SATCHMO the contradicted input clause instances are not saved; the contradiction only causes backtracking in the search for a proof.

2.3 Model Literal Replacement

Lee and Plaisted [5, 6] use *predicate replacement* to generate input clauses instances faster by partially hyper-linking literals in a clause. *Replace rules* are constructed from input clauses to apply predicate replacement. A replace rule is a clause

$$\{ \sim C_1, \sim C_2, \ldots, \sim C_n, L_1, L_2, \ldots, L_m \}$$

in the format of

$$C_1, C_2, \ldots, C_n \rightarrow L_1, L_2, \ldots, L_m$$

$C_i, 1 \leq i \leq n$, and $L_j, 1 \leq j \leq m$ could be positive or negative literals. Literals, $\sim C_1, \sim C_2, \ldots, \sim C_n$, to the left of the arrow are *distinguished literals*.

In predicate replacement, only distinguished literals in a replace rule are hyper-linked with literals from other clauses. This provides a faster way to generate instances using hyper-linking. Users can repeatedly apply predicate replacement indefinitely or just once. Predicate replacement has been used to prove many hard theorems in very short time (see [5]). It indicates that predicate replacement has a lot of potential in general theorem proving. However, often it requires deep understanding of the theorems to devise the replace rules. As a result, the replace rules are not natural and sometimes it is not easy to devise appropriate replace rules to obtain the proof fast.

We use a simplified predicate replacement; we only apply the replacement using the model literals and the idea is slightly different. The purpose is, instead of generating instances, to again check if the model causes any contradiction through predicate replacement.

The replace rules we use are *natural replace rules* and can be automatically generated from the input clauses. Thus human intervention is not needed. A replace rule is *natural* if

1. it has a unit consequence, namely, it has the format

$$C_1, C_2, \ldots, C_n \longrightarrow L$$

   where L and each C_i are literals.
2. each variable in L occurs in some C_i.

Once all C_i in a natural replace rule are hyper-linked with ground literals, L becomes ground after a proper substitution is applied. If every C_i is linked with a literal from a unit clause, we obtain a unit clause $\{ L\theta \}$ because all C_i are (unit) deleted. In this case, we can think of the replacement as deriving a unit consequence $L\theta$ from single facts $C_i\theta$. We use input clauses to generate all possible natural replace rules.

Here is how to use natural replace rules: we think of the literals in the model as literals from (ground) unit clauses (called *unit literals*) which can be seen as single facts. We then use the natural replace rules to generate more unit literals which can be seen as unit consequences or derived single facts. New unit literals are checked if there is any contradiction with old ones. We can repeat the

replacement more than once, just like the repeated replace rules used in [6, 5]. However, repeated natural replacement may not terminate. For example, the natural replace rule $p(X) \longrightarrow p(f(X))$ might keep generating larger and larger $p(f(f(\ldots)))$ literal and will not terminate. A bound on the number of replacements or the number of the derived unit literals should be used to guarantee termination. The natural replacement procedure is described in Fig. 3.

```
Algorithm Natural_Predicate_Replacement(M, Replace, Bound, S)
  Input
    M: set of literals in the model
    Replace: set of natural replace rules
    Bound: bound on the number of replacement or the derived literals

  Output
    S: set of literals in M, that cause contradiction

  Return Values
    FALSE: if no contradiction is found
    TRUE: if a contradiction is found

  begin
    {S_unit is a set of unit literals}
    S_unit := M
    while Bound is not exceeded do
      for each natural replace rule R = C_1, ..., C_n --> L in Replace do
        for each L obtained by hyper-linking all C_i with literals in S_unit do
          if L contradicts any literal in S_unit then
            S := { literals in M that derive L and ~L }
            return TRUE
          else
            S_unit := S_unit ∪ { L }
    return FALSE
  end
```

Fig. 3. Algorithm:*Natural_Predicate_Replacement*

For each derived literal L, we maintain a set of model literals which generate L using natural replacement. A contradiction is found if a newly derived literal L contradicts an existing literal $\sim L$. Suppose the L depends on the set S_L of model literals; $\sim L$ depends on the set S_{NL}. The algorithm returns $S = S_L \cup S_{NL}$. Then we add a clause $C = \{ M : \sim M \in S \}$ to the ground clause set. It is justified to add C as a ground clause because C is a logical consequence of the input clauses. Any further contradiction by C would also contradict the input clauses. Thus C can be added to the ground clause set, and we not only change the model but also avoid generating any model later that would cause the same contradiction.

Let us look at a simple example. Suppose model literals L_1 and L_2 generate L using the natural replace rule $L_1, L_2 \longrightarrow L$, and L_3 and L_4 generate $\sim L$ using

the natural replace rule $L_3, L_4 \longrightarrow \sim L$. Then L depends on the model literals $\{ L_1, L_2 \} = S_L$, and $\sim L$ depends on $\{ L_3, L_4 \} = S_{NL}$. Since L and $\sim L$ cause a contradiction, the algorithm returns a set $S = S_L \cup S_{NL} = \{ L_1, L_2, L_3, L_4 \}$. We then add a clause $C = \{ \sim L_1, \sim L_2, \sim L_3, \sim L_4 \}$ to the ground clause set. To illustrate (informally) that C is a logical consequence of the input clauses, we can think of C as a resolvent from resolving the two replace rules on L and $\sim L$. Since the two replace rules are actually input clauses, it easily follows that C is a logical consequence of the input clauses. C is added to the ground clause set to avoid generating any model that contains the subset $\{ L_1, L_2, L_3, L_4 \}$.

2.4 UR Resolution

Unit resolution [9] is a resolution involving one unit clause $U = \{ L \}$ and another clause C (which could also be a unit clause). Suppose C has a literal M and there exists a most general unifier θ such that $L\theta = \sim M\theta$. After applying unit resolution on U and C, the resolvent is $(C - M)\theta$. Since a unit clause has no other literals than the one resolved upon, the resolvent from unit resolution does not introduce new literals. This is compatible with the philosophy of hyper-linking that literals from different clauses should not be combined.

UR (*U*nit *R*esulting) resolution [10] is a sequence of unit resolutions in which the resolvent (called *UR resolvent*) is a unit clause. This can be seen as a multi-step unit resolution and is particularly helpful because the UR resolvents can then be used in further UR resolution.

We use UR resolution to detect a contradiction caused by the model found by model finding. The basic idea is to check if derived UR resolvents contradict any literal in the model. We keep generating new UR resolvents, which are unit clauses and using them in further UR resolutions. If a derived UR resolvent $\{ R \}$ contradicts a model literal L, that is, there exists a ground substitution θ such that $R\theta = \sim L$, we add $\{ R \}$ in the clause set (so no other model containing L can be generated) and look for another model.

A UR resolvent $\{ R \}$ is *relevant* if R can unify with a model literal L. Derived UR resolvents are kept temporarily to derive more UR resolvents. However, after UR resolution, we only keep the relevant UR resolvents; this heuristics allows us not to keep too many derived unit clauses and to save those resolvents relevant to the development of the proof.

UR resolution and the natural predicate replacement seem quite similar. The major difference is that we might save intermediate UR resolvents since they are logical consequences of input clauses. In natural predicate replacement we do not save any intermediate results because literals in the model are not really from unit clauses.

Like natural predicate replacement, UR resolution might not terminate. So a bound on the number of derived resolvents needs to be used.

3 Discussion

The search space of semantic hyper-linking grows when new literals are generated (for example, see Fig. 1 and 2). Also, in each round of semantic hyper-linking, ground clauses are generated to change the model (thus the new semantic structure) until the ground set is unsatisfiable [1]. We have described three model finding strategies that change the models without using semantics which is often expensive; at the same time, they do not increase the search space because they only generate clauses containing model literals or their negations.

Those refinements to incremental model finding still retain the completeness of semantic hyper-linking. They also maintain the soundness since they all generate ground logical consequences of the input clauses. More importantly, they have positive impact on the performance of semantic hyper-linking. In the proofs we have obtained so far, the model finding strategies almost always help.

To illustrate the power of the model finding strategies, let us look at two examples. IMV (intermediate value theorem in real analysis) and AM8 (attaining maximum theorem in real analysis) are two very hard theorems; these two hard theorems often generate very large search spaces that many powerful general theorem provers (for example, OTTER [8] and CLIN [5]) cannot handle. With proper semantics, semantic hyper-linking can prove IMV in 39 seconds using 4 rounds, and AM8 in 492 seconds using 7 rounds on a DEC5500. However, if the model finding strategies are not used, IMV will run over 10,000 seconds and still cannot find the proof; AM8 will take 2,500 seconds and 21 rounds.

	Filtering	Replacement	UR
Round 1	0	0	0
Round 2	0	1 (1)	2 (0)
Round 3	1 (1)	0	0
Round 4	3 (2)	2 (1)	0
Total	4 (3)	3 (2)	2 (0)

(a) IMV proof

	Filtering	Replacement	UR
Round 1	0	0	0
Round 2	0	0	1
Round 3	0	0	0
Round 4	1	0	0
Round 5	3	0	7
Round 6	2	1	12
Round 7	11	0	1
Total	17	1	21

(b) AM8 proof

Fig. 4. Ground clauses generated by model finding

The proof of IMV is found by an unsatisfiable ground clause set containing 12 clauses. Figure 4(a) shows the numbers of clauses generated by each model finding strategy during the proof of IMV. Numbers in parentheses represent the clauses in the unsatisfiable set, namely the useful clauses for the proof. Out of 12 clauses in the unsatisfiable set, 5 of them are generated by the model finding

strategies which generate in total 9 ground clauses.

The proof of AM8 is found by UR resolution of the model finding in round 7. Figure 4(b) shows the numbers of ground clauses generated by the model finding.

References

1. Heng Chu and David A. Plaisted. Semantically guided first order theorem proving using hyper-linking, 1992. Submitted.
2. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
3. P. C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM J. Res. Dev.*, pages 28–35, 1960.
4. J. Herbrand. Researches in the theory of demonstration. In J. van Heijenoort, editor, *From Frege to Gödel: a source book in Mathematical Logic, 1879-1931*, pages 525–581. Harvard Univ. Press, 1974.
5. Shie-Jue Lee. *CLIN: An Automated Reasoning System Using Clause Linking*. PhD thesis, University of North Carolina at Chapel Hill, 1990.
6. Shie-Jue Lee and David. A. Plaisted. Eliminating duplication with the hyper-linking strategy. *J. Automated Reasoning*, 9:25–42, 1992.
7. Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *Proc. of CADE-9*, pages 415–434, Argonne, IL, 1988.
8. William W. McCune. *OTTER 2.0 Users Guide*. Argonne National Laboratory, Argonne, Illinois, March 1990.
9. L. Wos, D. Carson, and G. Robinson. The unit preference strategy in theorem proving. In *Proceedings of the AFIPS Conference 26*, pages 615–621, Washington, D.C., 1964. Spartan Books.
10. Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

An Expressive Three-valued Logic with Two Negations

Douglas R. Busch*
DSV, Stockholm University and IDA, Linköping University

February 17, 1993

Abstract

Abstract This paper presents a flexible, expressive system $K_3^{\sim}$ of three-valued logic with *two* types of negation, having a *sequent* axiomatization which is an extension of the kind originally presented for Kleene's strong three-valued logic by Wang. The system $K_3^{\sim}$ turns out to be closely related to Lukasiewicz's three-valued logic. Applications: (1) Erik Sandewall has recently formulated a non-monotonic variant of three-valued logic. The non-monotonic "entailment" relation of his system can be expressed by a kind of "circumscription" formula in $K_3^{\sim}$. (2) J. Shepherdson has suggested that a hybrid three-valued intuitionistic logic could be useful in connection with Kunen's modification of Fitting's three-valued version of the Clark Completion semantics for logic programs with negation. A suitable "intuitionistic fragment" of $K_3^{\sim}$ is obtained by allowing in proofs only sequents with at most one formula in the succedent.

1 Axiomatizing three-valued logic

1.1 Kleene's strong three-valued logic

Kleene's strong three-valued logic [6] is characterized by the three-valued truth-tables:

	$\neg$
t	f
f	t
u	u

$\wedge$	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

$\vee$	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

$\rightarrow_K$	t	f	u
t	t	f	u
f	t	t	t
u	t	u	u

The implication symbol is subscripted to distinguish it from some other competing three-valued notions of implication, notably Lukasiewicz implication.

*Supported by Swedish agencies STU, STUF and TFR.

Kleene's logic has an axiomatization, but it is a *sequent* axiomatization. This avoids the problem that there are no "3-tautologies" to axiomatize, by axiomatizing instead the notion of *entailment* or *semantic consequence*, in a suitable three-valued sense, between formulas[1].

To be precise, we define K_3 to be the set of sequents $\Gamma \Rightarrow \Delta$, containing formulas built up in the usual way using $\neg$, $\wedge$ and $\vee$ satisfying $\Gamma \models_3 \Delta$, where this means that every 3-assignment which makes every formula in Γ take the value **t**, also makes at least one formula in Δ take the value **t**[2].

Now there are no "3-tautologies" , in Kleene's strong three-valued logic, not even $p \rightarrow_K p$, but there are some significant 3-entailments, for example:

$$p \models_3 p, \quad p, q \models_3 q, \qquad p \models_3 p, q.$$

1.2 A cut-free axiomatization

For classical logic it is well-known that Beth's *tableau method* and its notational variants in terms of semantic trees and the like are, in a sense, equivalent to Gentzen's sequent calculus [14].

Van Benthem [16] sketches how to modify the Beth tableau method so that it still works in the 3-valued or partial context. From the modified tableau method one can read off an axiomatization of K_3. The version presented here is further modified (slightly) to ensure *invertibility* with respect to the Kleene semantics.

We now interpret the right-hand side of a Beth tableau as "not true" , i.e. **f** or **u**. The usual reduction rules for conjunction and disjunction still work, and they are retained unchanged. *Both* of the usual reduction steps for negation are dropped. These correspond to the sequent rules:

$$\frac{\Gamma, \alpha \Rightarrow \Delta}{\Gamma \Rightarrow \neg\alpha, \Delta} \qquad \frac{\Gamma \Rightarrow \alpha, \Delta}{\Gamma, \neg\alpha \Rightarrow \Delta}$$

The reduction step that converts a negated $\neg\alpha$ on the right to an un-negated α on the left corresponds to the first of these, the rule $\Rightarrow \neg$. This is not even sound with respect to the three-valued semantics: consider $p \Rightarrow p$ and $\Rightarrow p, \neg p$. The dual Beth reduction step converts a negated $\neg\alpha$ on the left to un-negated α on the right. It corresponds to the sequent rule $\neg \Rightarrow$, which is sound, but not invertible with respect to the Kleene semantics: consider $\Rightarrow p, \neg p$ and $\neg\neg p \Rightarrow p$

When the Beth tableau approach is applied in the context of classical two-valued logic these two rules for negation ensure that all negation signs are eventually eliminated: a negated formula on one side sheds its negation sign and moves over to the other side of the tableau. Here in three-valued or partial logic, on our approach, this *never* happens. Negated formulas, either on the left or the

[1] According to Feferman [4], the first presentation of such a sequent axiomatization of Kleene's logic is due to Wang [17], but the basic idea has been re-discovered several times.

[2] This notion of entailment,which is a direct simple-minded generalization of the ordinary notion of semantic consequence in two-valued logic, has evidently [16] been independently formulated by Hans Kamp under the name "strong consequence".

right side of a tableau, are simplified by having the negations "pushed inwards" until they apply only to atoms. This is brought about by using Double-Negation Elimination and De Morgan's Laws as re-write rules:

$$\neg\neg\alpha \mapsto \alpha \qquad \neg(\alpha \wedge \beta) \mapsto \neg\alpha \vee \neg\beta \quad \neg(\alpha \vee \beta) \mapsto \neg\alpha \wedge \neg\beta \quad .$$

The "atomic negation" approach: It is possible to build it into the definition of *formula* for three-valued logic that negation applies only to atoms. We can define a *reduced formula* to be one built up from *literals* in the usual way using $\wedge$ and $\vee$, then view what would ordinarily be the formulas as "syntactic sugar" standing for reduced formulas. Now the steps driving $\neg$ inwards are not inferences, they are formally just part of the manipulation of the syntax. They can all be performed before running the tableau procedure, or they can be interspersed among tableau reduction steps.

The criterion for closure of a (sub-)tableau needs to be modified. Now a (sub-)tableau closes if it contains the same *literal* (atom or negated atom) on each side, or if it contains a complementary pair of literals $p, \neg p$ on the *left*.

PROPOSITION: $\Gamma \Rightarrow \Delta \in K_3$ iff a tableau starting with Γ on the left and Δ on the right closes, equivalently iff $\Gamma \Rightarrow \Delta$ is provable from the following sequent calculus postulates.

Axiomsfor p an atom:

$$p \Rightarrow p \qquad \neg p \Rightarrow \neg p \qquad p, \neg p \Rightarrow$$

Introduction Rules: the usual rules for conjunction and disjunction:

$$\frac{\Gamma, \phi, \psi \Rightarrow \Delta}{\Gamma, \phi \wedge \psi, \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow \Delta, \phi \quad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \phi \wedge \psi}$$

$$\frac{\Gamma, \phi \Rightarrow \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \phi \vee \psi \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow \phi, \psi, \Delta}{\Gamma \Rightarrow \phi \vee \psi, \Delta}$$

Structural Rules: Weakening and Cut:

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma, \Phi \Rightarrow \Psi, \Delta} \qquad \frac{\Gamma, \alpha \Rightarrow \Delta \qquad \Gamma \Rightarrow \alpha, \Delta}{\Gamma \Rightarrow \Delta}$$

Cut Elimination holds, as for classical logic. Any sequent which is provable using Cut also has a "direct" proof that does not use Cut.

Special rules for negation Instead of incorporating the treatment of negation into the syntax, we can get the same effect by postulating six special rules for $\neg$:

$$\frac{\Gamma, \alpha \Rightarrow \Delta}{\Gamma, \neg\neg\alpha \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow \alpha, \Delta}{\Gamma \Rightarrow \neg\neg\alpha, \Delta}$$

$$\frac{\Gamma, \neg\alpha \Rightarrow \Delta \quad \Gamma, \neg\beta \Rightarrow \Delta}{\Gamma, \neg(\alpha \wedge \beta) \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow \neg\alpha, \neg\beta, \Delta}{\Gamma \Rightarrow \neg(\alpha \wedge \beta), \Delta}$$

$$\frac{\Gamma, \neg\alpha, \neg\beta \Rightarrow \Delta}{\Gamma, \neg(\alpha \vee \beta) \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow \neg\alpha, \Delta \quad \Gamma \Rightarrow \neg\beta, \Delta}{\Gamma \Rightarrow \neg(\alpha \vee \beta)}$$

1.3 Adding external negation

There is another notion of negation which arises naturally in the three-valued context, the *external* negation $\sim \alpha$, defined to be **t** if α is **f** or **u**, and **f** if α is **t**. It is easy to check that $\phi, \alpha \models_3 \psi$ holds iff $\phi \models_3 \sim \alpha, \psi$ and also $\phi, \sim \alpha \models_3 \psi$ holds iff $\phi \models_3 \alpha, \psi$. Slightly more generally, we can see that the *usual* sequent conditions for negation, namely:

$$\frac{\Gamma \Rightarrow \alpha, \Delta}{\Gamma, \sim \alpha \Rightarrow \Delta} \qquad \frac{\Gamma, \alpha \Rightarrow \Delta}{\Gamma \Rightarrow \sim \alpha, \Delta}$$

apply to this *external* negation. That is, these rules are sound when $\Rightarrow$ is interpreted as $\models_3$, and they are also invertible with respect to the strong Kleene semantics.

PROPOSITION: In fact these two introduction rules extend the previously mentioned complete axiomatization of K_3 to a complete axiomatization of $K_3^{\sim}$.

If we want to take the "reduced formula" approach, so that $\neg$ strictly applies only to atoms, we have to stipulate how to interpret $\neg \sim \alpha$, since now reduced formulas are built from ($\neg$-)literals using $\wedge$, $\vee$ and $\sim$. We stipulate:

$$\neg \sim \alpha :=\sim\sim \alpha$$

1.4 A natural implication

Once the external negation is available, it is natural to define

$$\alpha \supset \beta :=\sim \alpha \vee \beta.$$

[3] Unlike $\rightarrow_K$, this notion of implication satisfies the "deduction theorem" relationship:

$$\Gamma, \alpha \models_3 \beta \qquad \text{holds iff} \qquad \Gamma \models_3 \alpha \supset \beta.$$

More generally, the *usual* sequent rules for implication hold with respect to this notion of implication:

$$\frac{\Gamma, \beta \Rightarrow \Delta \qquad \Gamma \Rightarrow \alpha, \Delta}{\Gamma, \alpha \supset \beta \Rightarrow \Delta} \qquad \frac{\Gamma, \alpha \Rightarrow \beta, \Delta}{\Gamma \Rightarrow \alpha \supset \beta, \Delta}$$

are both sound for $\models_3$. This notion of implication is the weakest one for which this holds.

2 Łukasiewicz's three-valued logic

Kleene's three-valued tables differ only slightly from tables earlier presented by Lukasiewicz, as Kleene himself points out [6]. The tables for negation, conjunction and disjunction are identical, and so is the table for implication, except for just *one* case, the case where both inputs are **u**. Whereas Kleene's strong

[3] This notion of implication has also been employed by Schmitt [11].

table for implication makes $\mathbf{u} \rightarrow_K \mathbf{u} = \mathbf{u}$, for Lukasiewicz implication we have instead $\mathbf{u} \rightarrow_L \mathbf{u} = \mathbf{t}$. Lukasiewicz's system also contains unary operators **L** and **M**, intended as formalizing some kind of temporal necessity and possibility. The tables are:

$\rightarrow_L$	t	f	u
t	t	f	u
f	t	t	t
u	t	u	t

$\leftrightarrow_L$	t	f	u
t	t	f	u
f	f	t	u
u	u	u	t

α	Lα	Mα
t	t	t
f	f	f
u	f	t

In contrast to Kleene's logic, Lukasiewicz's system does contain some 3-tautologies, for example $p \rightarrow_L p$. The price for this is that the Lukasiewicz implication is not "regular" in Kleene's sense, a sort of non-monotonicity . **PROPOSITION**: The Lukasiewicz implication is definable within $K_3^{\sim}$:

$$\alpha \rightarrow_L \beta := (\alpha \supset \beta) \wedge (\neg\beta \supset \neg\alpha).$$

Thus $\rightarrow_L$ can be viewed as built up in two steps within $K_3^{\sim}$.

1. First, $\alpha \supset \beta$ i.e. $\sim \alpha \vee \beta$ is the weakest implication $\rightarrow$ given by a three-valued truth-table satisfying:

$$\models_3 \alpha \rightarrow \beta \qquad \text{iff} \qquad \alpha \models_3 \beta.$$

2. Now to get an implication which satisfies contraposition,

$$\models_3 \alpha \rightarrow \beta \qquad \text{implies} \qquad \models_3 \neg\beta \rightarrow \neg\alpha,$$

we take $(\alpha \supset \beta) \wedge (\neg\beta \supset \neg\alpha)$.

This shows that the Lukasiewicz system L_3 is by no means as arbitrary and un-motivated as has been commonly alleged, for example by Urquhart [15] and Feferman [4].

3 Sandewall's non-monotonic three-valued logic

A new very interesting *non-monotonic* three-valued logic has recently been formulated by Erik Sandewall [10]. Perhaps its most interesting feature is the explicit default operator **D**. Dα is intended to represent that α has been assumed by default, but **D** is *not* 3-truth-functional. Sandewall presents a kind of intensional semantics for this operator, which has subsequently been modified and developed by Doherty and Lukaszewicz [2]. As well as the **D**-operator, Sandewall defines "external operators" **L**, **M** and **N** determined by the tables:

α	Lα	Mα	Nα
t	t	t	f
f	f	f	f
u	f	t	t

Sandewall's **L** and **M** are actually the same as Lukasiewicz's, and they are definable from $\sim$, $\neg$ and $\wedge$:

$$\mathbf{L}\alpha := \neg \sim \alpha \qquad \mathbf{M}\alpha :=\sim \neg\alpha \qquad \mathbf{N}\alpha :=\sim \alpha \wedge \sim \neg\alpha$$

3.1 A non-monotonic entailment relation

Instead of using the relation $\models_3$ as an entailment relation, Sandewall uses a *non-monotonic* modification of it, after the style of Shoham [13]. We write:

$$\Gamma \models_3^* \beta$$

to express that β is **t** in every $\sqsubseteq$-minimal 3-model of Γ. Here $\sqsubseteq$ is the *information ordering* between 3-valuations, i.e. if 3-valuations or partial valuations are coded in the obvious way as sets of literals (atoms or negated atoms), the information ordering is just set inclusion. That is, we don't consider *all* the 3-models of Γ, we restrict ourselves to those that are "as undefined as possible". For example, if p, q are distinct atoms,

$$p \models_3^* \mathbf{N}q$$

holds. In the *minimal* 3-model for p we have $p = \mathbf{t}$ and $q = \mathbf{u}$.

Now we sketch how this non-monotonic relation $\models_3^*$ can be expressed in $K_3^{\sim}$ by a kind of circumscription formula. We want to find for each formula α another formula $\mathcal{C}(\alpha)$, with the property that its models are precisely the minimal models of α . Then we will have the equivalence:

$$\alpha \models_3^* \beta \qquad \text{iff} \qquad \mathcal{C}(\alpha) \models_3 \beta.$$

To illustrate, let us suppose that $\alpha = \alpha(p, q, r)$ contains just the three variables p, q, r. Suppose that a particular 3-assignment w is a model for α. Then this assignment is a *minimal* model for α iff it is "as undefined as possible": none of the 3-assignments which have more cases of **u** than this one , is a model. Now look at each atom separately. Consider the value w assigns to p: if it is **u**, then it cannot be made informationally less. But if it is defined, i.e. either **t** or **f**, then if this value is changed to **u**, the resulting assignment will no longer satisfy α — *if* indeed the model w is minimal.

Now we can test whether p is defined by the formula $p \vee \neg p$, and we can express that the result of replacing p by **u** in α is not a model by $\sim \alpha(\mathbf{u}, q, r)$

That is

$$(p \vee \neg p) \supset \sim \alpha(\mathbf{u}, q, r)$$

Call this $\mathcal{M}_p(\alpha)$, and likewise define

$$\mathcal{M}_q(\alpha) := (q \vee \neg q) \supset \sim \alpha(p, \mathbf{u}, r)$$

and

$$\mathcal{M}_{pq}(\alpha) := [(p \vee \neg p) \wedge (q \vee \neg q)] \supset \sim \alpha(\mathbf{u}, \mathbf{u}, r).$$

We obtain the strengthening of $\mathcal{C}(\alpha)$ we want, to a $\mathcal{C}(\alpha)$ which has as its models just the *minimal* models of α, by taking

$$\mathcal{C}(\alpha) := \alpha \wedge \mathcal{M}_p(\alpha) \wedge \mathcal{M}_q(\alpha) \wedge \mathcal{M}_r(\alpha) \wedge \mathcal{M}_{qr}(\alpha) \wedge \mathcal{M}_{pr}(\alpha) \wedge \mathcal{M}_{pq}(\alpha) \wedge \mathcal{M}_{pqr}(\alpha).$$

Finally[4], to reduce $\alpha \models_3^* \beta$ let $\vec{q}$ be the list of atoms which occur in β,but do not occur in α. Then these should all get the value **u** in a minimal model, so we can take the translation to be

$$\mathcal{C}(\alpha) \wedge \mathbf{N}\vec{q} \models_3 \beta.$$

4 Semantics for logic programs with negation

Shepherdson[5] remarks that it would be desirable somehow to combine three-valued with *intuitionistic logic*, and he suggests that one might do this starting from a "complete and consistent deductive system for 3-valued logic, as Ebbinghaus has done for a very similar kind of 3-valued logic", and he cites Ebbinghaus's paper [3].[6]

A hybrid logic of the kind he envisages could then be used to "tighten" further Kunen's [7] modification of Fitting's [5] three-valued semantics for logic programs with negation as failure, in order to get a closer fit between the three-valued declarative semantics of the (Clark completion of) a logic program with negation, and the procedural semantics given by SLDNF resolution.

Kunen's semantics matches the behaviour of SLDNF-resolution rather well, and for *propositional* programs it is complete. But there are cases where the semantics is too strong, supporting answers where nothing resembling PROLOG would. Kunen [9] gives this example:

$$\begin{array}{rcl} p & \leftarrow & \neg q(X) \\ q(c) & \leftarrow & \\ q(X) & \leftarrow & \neg r(X) \\ r(c) & \leftarrow & \end{array}$$

In the Clark completion $\neg p$ is equivalent to $\forall X(X = c \vee X \neq c)$. This is always **t** in Kunen's semantics, so the semantics has to support a *yes* answer. What is more, as Kunen points out, the program itself is sufficiently well-behaved (*hierarchical* and *strict*) that the three-valued semantics for it reduces to the two-valued semantics. Yet SLDNF-resolution can't derive indefinite answers, so here it won't agree with the semantics.

[4] This treatment presupposes of course that **u** is a propositional constant in the language. But if it is not, just take a new atom v not mentioned in α or β, add to α the condition $\mathbf{N}v$,and treat v as though it were **u**.

[5] We refer to Shepherdson's survey article [12] for any terminology not explained here, and to the articles by Kunen [7, 8, 9] and Fitting [5].

[6] Shepherdson must have been unaware of the axiomatizations of K_3 that exist in the literature going back to Wang [17], even though Wang's paper is cited by Ebbinghaus.

Can the semantics be weakened? Kunen raises the possibility of letting $=$ be interpreted as any three-valued relation which makes the equality axioms in CET come out as **t**. Then there could be a model containing an element a such that $a = c$ is **u**, and then $r(a)$, $q(a)$ and p would be **u** too. But the problem with this is that $r(a) \wedge \neg r(a)$ becomes **u** as well, so failure is *not* supported for the query $? - r(X), \neg r(X)$, and SDLNF-resolution would not be sound with respect to this weakened semantics.

Kunen also addresses Shepherdson's suggestion to somehow bring in intuitionistic logic. Shepherdson had shown that the original statement of soundness of SLDNF-resolution with respect to the 2-valued consequences of *Comp(P)* could be strenthened to the stricter notion of intuitionistic consequence, i.e. derivability: if a query $? - \lambda_1, \ldots, \lambda_r$ succeeds with answer θ, then $Comp(P) \vdash_I \forall[(\lambda_1 \wedge \ldots \wedge \lambda_r)\theta]$, and if it fails, then $Comp(P) \vdash_I \forall[\neg(\lambda_1 \wedge \ldots \wedge \lambda_r)]$.

Kunen points out that we can't just use intuitionistic proof theory in a semantics for SLDNF-resolution, because from the program

$$p \leftarrow p$$

an answer *no* to the query $? - p, \neg p$ should not be supported, whereas a semantics that took over all of intuionistic logic would have to support this answer, as $\vdash_I \neg(p \wedge \neg p)$. The problem here is that this is exactly the sort of example that called for three-valued logic in the first place. The PROLOG interpreter will loop, and this is represented by setting p to **u**.

On the other hand , to go back to Kunen's first example, $\vdash_I$ does seem to be exactly what is called for here, to represent the fact that that the interpreter can't derive the indefinite conclusion $\forall X(X = c \vee X \neq c)$. Actually many people have remarked that PROLOG behaves rather intuitionistically. If the clauses are viewed directly as sequents in the obvious way, even though the logic is supposed to be classical, everything is happening in the intuitionistic fragment, since (definite) clauses are a particular case of intuitionistic sequents, i.e. with a single formula, actually an atom, in the succedent.

4.1 intuitionistic three-valued logic

The following system seems to be a reasonable way of carrying out Shepherdson's proposal to amalgamate intuitionistic and three-valued logic to get " $\vdash_{3I}$". We start with the full version of the sequent calculus for first-order logic.[7] To be definite, let us take it to be the version in Kleene's treatise [6].

- To make the system intuitionistic, we impose the restriction that all sequents in proofs have at most one formula in the succedent.
- To make the system three-valued, we ban the use of $\Rightarrow \neg : \frac{\Gamma, \alpha \Rightarrow \Delta}{\Gamma \Rightarrow \neg\alpha, \Delta}$.
- To offset the loss of $\Rightarrow \neg$ (partially), we retain those of the special rules for negation previously formulated that are intuitionistically correct as well.(Details omitted here because of space limitations.)

[7] The three-valued treatment sketched so far extends straightforwardly to predicate logic.

- *Optionally* we can postulate that double negation elimination holds for *atoms* by adding an axiom $\neg\neg p(\vec{x}) \Rightarrow p(\vec{x})$ for each atomic predicate $p(\vec{x})$.

This specifies the inference mechanism. My modified version of Kunen's semantics uses these rules for the "intuitionistic fragment" of K_3 . The semantics is still recursively enumerable and $Comp(P)$ is essentially unchanged, but we emphasize that:

1. The constraint that $=$ is to be two-valued has to be expressed by the sequent
   $\Rightarrow \neg(X = Y \wedge X \neq Y)$, and *not* by $\Rightarrow X = Y \vee X \neq Y$. In the context of intuitionistic logic the disjunction has a distinctive meaning, such that we can't affirm a disjunction unless we are in a position to affirm one or other of the disjuncts, and this notion would be too strong for what we want to say.

2. We interpret the disjunction in the completed definitions in $Comp(P)$ as the intuitionistic disjunction.

This tightening of Kunen's semantics evidently satisfies soundness, and it matches the procedural behaviour of SLDNF-resolution better. Going back to Kunen's example, we can see that

1. A *yes* answer to the query $? - \neg p$ is now *not* supported, since we know $\Rightarrow X = Y \vee X \neq Y$ is not derivable using $\vdash_{3I}$ from *Comp(P)*, not even from $\Rightarrow \neg(X = Y \wedge X \neq Y)$. We know this because $X = Y \vee X \neq Y$ is underivable even with full intuitionistic logic $\vdash_I$, and intuitionistically also
   $\neg(X = Y \wedge X \neq Y)$ is an instance of a logical law.

2. Failure is supported for the query $? - r(X), \neg r(X)$, because $r(X)$ is equivalent to $X = c$ and $\Rightarrow \neg(X = c \wedge X \neq c)$ is an instance of our postulated $\Rightarrow \neg(X = Y \wedge X \neq Y)$.

3. On the other hand, we are *not* postulating $\Rightarrow \neg(p \wedge \neg p)$ for all other atoms p or for all formulas, *only* for the equality predicate $X = Y$. We know that $\Rightarrow \neg(p \wedge \neg p)$ is not derivable, because it isn't derivable in K_3 . Thus we *don't* get a problem with the query $? - p, \neg p$ addressed to the program $p \leftarrow p$.

References

[1] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases.* Plenum, New York, 1978.

[2] P. Doherty. *NML3 - A Non-Monotonic Formalism with Explicit Defaults.* PhD thesis, Linköping University, Sweden, 1991.

[3] H.-D. Ebbinghaus. Über eine Prädikatenlogik mit partiell definierten Prädikaten und Funktionen. *Arch. math. Logik*, 12:39–53, 1969.

[4] S. Feferman. Towards useful type-free theories I. *Journal of Symbolic Logic*, 49:75–111, 1984.

[5] M. C. Fitting. A Kripke/Kleene semantics for logic programs. *J. Logic Programming*, pages 295–312, 1985.

[6] S. C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand Company Inc., Princeton, New Jersey, 1952.

[7] K. Kunen. Negation in logic programming. *J. Logic Programming*, pages 289–308, 1987.

[8] K. Kunen. Some remarks on the competed database. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, volume 2, pages 978–992, Cambridge, Massachusetts and London, England, 1988. MIT Press.

[9] K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.

[10] E. Sandewall. The semantics of non-monotonic entailment defined using partial interpretations. In M. Reinfrank, J. de Kleer, M. Ginsberg, and E. Sandewall, editors, *Non-Monotonic Reasoning 2nd International Workshop, Grassau, FRG June 1988, Proceedings*, volume 346 of *Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science, Edited by J. Siekmann , No 346 .*, pages 27–41, Berlin Heidelberg New York, 1989. Springer Verlag.

[11] P. H. Schmitt. Computational aspects of three valued logic. In *Proc. 8th Conf. Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 190–198. Springer-Verlag, 1986.

[12] J. C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, Los Altos, California, 1988. Previously available as Report PM-01-87, School of Mathematics, University of Bristol.

[13] Y. Shoham. *Reasoning about Change*. MIT Press, 1988.

[14] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

[15] A. Urquhart. Many-valued logic. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume III, pages 71–116. D. Reidel, 1986.

[16] J. van Benthem. *A Manual of Intensional Logic*. CSLI, Standford University, California 94305, 2nd edition, 1988.

[17] H. Wang. The calculus of partial predicates and its extension to set theory I. *Zeitschr. math. Logik Grundl. Math.*, 7:283–288, 1961.

Compiling Proof Search in Semantic Tableaux

Joachim Posegga

Universität Karlsruhe
Institut für Logik, Komplexität und Deduktionssysteme
Am Fasanengarten 5, 7500 Karlsruhe, FRG
posegga@ira.uka.de

February 8, 1993

Abstract

An approach to implementing deduction systems based on semantic tableaux is described; it works by *compiling* a graphical representation of a fully expanded tableaux into a program that performs the search for a proof at runtime. This results in more efficient proof search, since the tableau needs not to be expanded any more, but the proof *consists of determining whether it can* be closed, only. It is shown how the method can be applied for compiling to the target language Prolog, although any other general purpose language can be used.

1 Introduction

The basic idea of tableau–based systems (see (Fitting, 1990) for a good introduction) is to try to prove inconsistency of a formula by failure of a systematic model–construction process: it is tried to satisfy a formula by stepwise refinement of potential models, and a proof is found if all those models can be ruled out by detecting contradictions in them. This working principle became more and more popular in automated deduction during the last years, after resolution has governed the field for two decades. Essentially, the former proof procedures offer a closer relation to semantics than resolution does. This is handy if a deduction system is supposed to be integrated into an application, rather than designed as a stand–alone prover. The interest in this steadily increases, which explains the shift of interest.

In order to apply deduction techniques, a concrete implementation method is required: the standard way of implementing tableaux-based provers is to keep an explicit datastructure representing a tableau (usually as a set of its branches) and modify it during runtime. This can be compared with an interpreter for a programming language: the tableau is a program containing statements to be executed (i.e.: formulæ to be expanded), until certain conditions are met and the program terminates (i.e.: all branches are closed). This paper shows how the proof search can be speeded up by compilation techniques. The basic idea is to compile a fully expanded tableau into a program that carries out the proof search at runtime.

The underlying idea is derived from the author's work on compilation techniques for first–order deduction with Shannon graphs (Posegga & Ludäscher, 1992; Posegga, forthcoming 1993) and works as follows: First, an arbitrary first–order formula is transformed into a graphical representation of a fully expanded tableau for it. Then, the graph is compiled into a program which shows the formulæ's inconsistency when it is executed. The execution reflects the proof search in semantic tableaux and tries to close every branch in the tableau. We will show how the principle works for Prolog as target language, although any other general-purpose language can be used.

The advantage of our approach is that some of the effort for the proof search (namely expanding the tableau) can be moved to a preprocessing phase that derives the graph and generates the program for it. This preprocessing is of only linear complexity in time and space w.r.t. the length of the negation normal form of the input formula. This is due to the fact that a graph instead of a tree is used for representing the fully expanded tableau; it uses structure sharing and represents multiple occurrences of subtrees in a tableau only once.

Note, that there is only a notational difference between a tableau represented as a graph and as a tree. From a theoretical point of view, trees are much handier than graphs, since we can use a linear notation and regard them simply as logical formulæ. We will refer to trees for the theoretical treatment of our method, but the reader should keep in mind that an implementation should use a graphical representation.

The paper is written from a practical point of view and assumes to be familiar with the theoretical background of semantic tableaux. When arguing on the implementation level, clearness and readability is preferred over showing how to achieve efficient code. The paper starts by discussing compilation techniques for propositional formulæ in Section 2. The framework is carried forward to the first–order level in Section 3; Chapter 4 draws conclusions from our research.

2 Propositional Logic

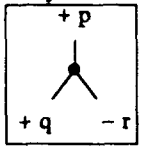
For reasons becoming clear soon, we will use a slightly unusual notation for writing down tableaux and consider them simply as logical formulæ with signed atoms; "+" and "−" serve as signs. Let $\mathcal{L}^0$ be the language of propositional calculus defined in the usual way, and $\mathcal{L}^0_{At}$ the atomic formulæ of $\mathcal{L}^0$.

Definition 2.1 (Set of Fully Expanded Propositional Tableaux)
The set $\mathcal{T}ab$ *of fully expanded propositional tableaux is defined to be the smallest set such that*

1. $\mathbf{1} \in \mathcal{T}ab$ *("1" denotes the atomic truth value "true")*
2. *if* $\mathcal{T} \in \mathcal{T}ab$ *and* $A \in \mathcal{L}^0_{At}$, *then* $(+A)\wedge\mathcal{T}$ *and* $(-A)\wedge\mathcal{T} \in \mathcal{T}ab$.
3. *if* $\mathcal{T}_1, \mathcal{T}_2 \in \mathcal{T}ab$, *then* $\mathcal{T}_1 \vee \mathcal{T}_2 \in \mathcal{T}ab$.

The elements of $\mathcal{T}ab$ *will be denoted by letters of the calligraphic alphabet* "$\mathcal{A}, \mathcal{B}, \ldots$".

The intuition behind this notation is that the formulæ on a branch denote a conjunction, and that branching means disjunction. As a simple example, consider the formula "$p\wedge(q\vee\neg r)$"; assume we start a tableau with this formula and expand it completely in the standard way.

We get: 

which can be written as $+p\wedge((+q\wedge 1)\vee(-r\wedge 1))$, an element of $\mathcal{T}ab$.

The atom "1" is superfluous, but handy, as we will see soon. It can be regarded as a mark for the end of a branch.

Next, we will see how such a representation can be derived for a formula. The basic idea is to recursively compute fully expanded tableau for compound formulæ, and to combine these according to the logical connectives. The following notation will be needed for handling conjunctions:

Definition 2.2 (Replacement of 1–nodes)
Let $\mathcal{A}, \mathcal{B} \in \mathcal{T}ab$; the replacement of 1*–nodes in $\mathcal{A}$ by $\mathcal{B}$ is recursively defined as:*

$$\mathcal{A}\left[\frac{1}{\mathcal{B}}\right] = \begin{cases} \mathcal{B} & \text{if } \mathcal{A} = 1 \\ \mathcal{A}_1\wedge\left(\mathcal{A}_2\left[\frac{1}{\mathcal{B}}\right]\right) & \text{if } \mathcal{A} = \mathcal{A}_1\wedge\mathcal{A}_2 \\ \left(\mathcal{A}_1\left[\frac{1}{\mathcal{B}}\right]\right)\vee\left(\mathcal{A}_2\left[\frac{1}{\mathcal{B}}\right]\right) & \text{if } \mathcal{A} = \mathcal{A}_1\vee\mathcal{A}_2 \end{cases}$$

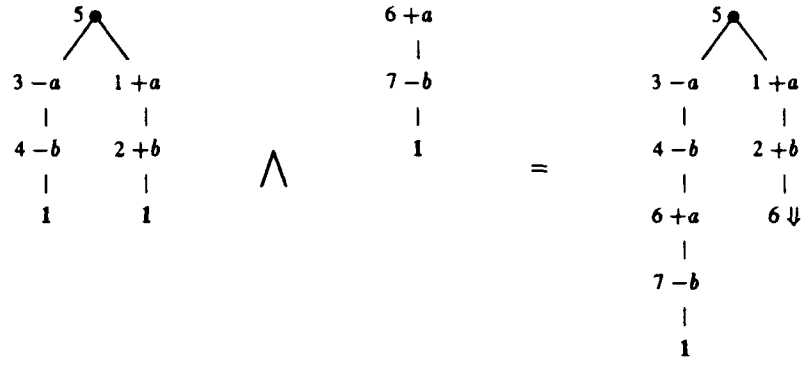
The following recursive function *conv* maps an arbitrary propositional formula to a fully expanded propositional tableaux for it:

Definition 2.3

$$conv(F) = \begin{cases} (+F)\wedge 1 & \text{if } F \in \mathcal{L}_{At} \\ (-A)\wedge 1 & \text{if } F = \neg A, A \in \mathcal{L}_{At} \\ conv(F') & \text{if } F = \neg\neg F' \\ conv(\alpha_1)\left[\frac{1}{conv(\alpha_2)}\right] & \text{if } F \text{ is a } \alpha\text{–formula} \\ conv(\beta_1)\vee conv(\beta_2) & \text{if } F \text{ is a } \beta\text{–formula} \end{cases}$$

If we perform the replacement operation in the above definition by replacing edges to nodes, rather than replacing nodes themselves, we can derive a directed, acyclic graph. One easily verifies that, in this case, computing the mapping is of linear complexity in time and space w.r.t. the length of the input formula in negation normal form.

As a simple example, assume we want to derive a graph representing a fully expanded tableau for $(a \leftrightarrow b)\wedge a\wedge\neg b$. First, we compute the graph for $(a \leftrightarrow b)$, which is an β–formula with $\beta_1 = \neg a\wedge\neg b$ and $\beta_2 = a\wedge b$. After computing the graph for $a\wedge\neg b$, we conjunctively combine both by replacing all *edges* to 1–nodes in the first graph by an edge to the second graph:



The numbers attached to each node in the graph will be needed later for the compilation. We will assume that the generated nodes are consecutively numbered, although it would be sufficient that they uniquely denote each node. The node "6 ⇓" is just an aid for drawing the graph and means that the edge leading to it actually leads to node number 6.

Besides the fact that the tableau is represented as a graph with numbered nodes, there is no difference to a standard semantic tableau. The above formula was inconsistent, so the tableaux represented by the right graph is closed, i.e.: each branch of the tableau is contradictory. In terms of the graph this means, that both paths from the root to the 1–leaf are contradictory. Once we have derived such a graph for a propositional formula, the only thing left to do for obtaining a proof is to test this condition.

This is exactly the idea of the proposed method for compiling the proof search: we compute the graph in the above way and compile it into a program that is procedurally equivalent to the above test. Any general–purpose target language can be used, but it is particularly easy to explain the process with Prolog:

As propositional logic is decidable, we can determine whether or not the graph we consider has an open path to a 1–leaf, and therefore the tableau has an open branch. If this is the case, we have derived a model for the formula. We will generate a program that enumerates all models, i.e., it enumerates all open paths in the graph. The method to achieve this is quite straightforward: for each node in the graph a Prolog clause `node/2` is generated that succeeds if an open path through this node to the 1–leaf exists. We will use the numbers of the graph nodes to distinguish the clauses for the nodes. This number is the first parameter of a `node`–clause[1], the second is the path that has been constructed to reach the node.

There are two types of nodes in a graph:

1. binary "∨"–nodes: in this case the clause for the node succeeds if an open path can be constructed through one of the successors.

[1] This is done for performance reasons, since Prolog systems usually perform indexing on the first argument of clauses.

2. unary "$\wedge$"–nodes labeled with a literal: here, the clause succeeds if the literal can be added to the path without yielding a contradiction, and an open path through the successor node exists. If the successor node is 1, the last condition is always true.

A minor technical problem to be solved is finding an efficient representation of paths: we easily can determine the number of *different* atoms in a formula during the construction of the graph, so a good solution is to use a Prolog–term of this length. Each argument in this term represents the truth value of the according atom (denoted by "+" or "-"). The following clauses "implement" the graph for the fully expanded tableau of $(a \leftrightarrow b) \wedge a \wedge \neg b$ above; the atom a appears at the first position in the path, and b at the second. It should be easy to see that `satisfy` succeeds with a path (a model) if there is an open branch in the tableau, and fails, otherwise:

```
node(5,Path)   :- (node(3,Path) ; node(1,Path)).
node(3,Path)   :- arg(1,Path,-), node(4,Path).
node(4,Path)   :- arg(2,Path,-), node(6,Path).
node(6,Path)   :- arg(1,Path,+), node(7,Path).
node(7,Path)   :- arg(2,Path,-).
node(1,Path)   :- arg(1,Path,+), node(2,Path).
node(2,Path)   :- arg(2,Path,+), node(6,Path).
satisfy(Path)  :- functor(Path,path,2),node(5,Path).
```

In the propositional case it is surprisingly easy to switch from Prolog to another target language. As we have seen, the only thing to do is to generate code that simulates descending in the graph and collects literals that form paths. This can be achieved in conventional programming languages by defining a function for each node that recursively calls functions for successor nodes after assigning a corresponding truth value to the atom of the node, if possible. Practical experiments with C and 8086 Assembler have shown, that Prolog programs of the above kind run roughly as fast as C, but about 20-30 times slower than Assembler.

3 First–order Logic

The basic difference in the proof search between propositional and first–order tableaux is that we must deal with γ–formulæ on branches in the latter case. Such formulæ have the peculiarity that they may be used more than once during the proof search. It is therefore not sufficient to determine a fully expanded tableau for a formula and try to show that it is closed. We will handle this by extending the definition of a fully expanded tableau, such that not only atoms, but also fully expanded tableaux for γ–formulæ can appear on branches. Such a node will be called a γ–node; the graph inside is called a γ–graph.

Let $\mathcal{L}$ be the language of first–order calculus and $\mathcal{L}_{At}$ the atomic formulæ of $\mathcal{L}$.

Definition 3.1 (Set of Fully Expanded First–order Tableaux)
The set $\mathfrak{Tab}_{\forall}$ of fully expanded first-order tableaux is defined to be the smallest set such that

1. $\mathfrak{Tab} \subset \mathfrak{Tab}_{\boxed{\forall}}$
2. *if* $\mathcal{T}_1, \mathcal{T}_2 \in \mathfrak{Tab}_{\boxed{\forall}}$ *and* $x_1, \ldots, x_n$ *are free variables in* $\mathcal{T}_1$, *then* $((\forall(x_1, \ldots, x_n)\, \mathcal{T}_1) \wedge \mathcal{T}_2) \in \mathfrak{Tab}$.

The first–order counterpart of *conv* will be denoted by $conv_{\boxed{\forall}}$ and is defined as follows:

Definition 3.2

$$conv_{\boxed{\forall}}(F) = \begin{cases} (+F)\wedge\mathbf{1} & \text{if } F \in \mathcal{L}_{At} \\ (-A)\wedge\mathbf{1} & \text{if } F = \neg A, A \in \mathcal{L}_{At} \\ conv_{\boxed{\forall}}(F') & \text{if } F = \neg\neg F' \\ conv_{\boxed{\forall}}(\alpha_1)\left[\frac{1}{conv_{\boxed{\forall}}(\alpha_2)}\right] & \text{if } F \text{ is a } \alpha\text{–formula} \\ \left(conv_{\boxed{\forall}}(\beta_1)\right) \vee \left(conv_{\boxed{\forall}}(\beta_2)\right) & \text{if } F \text{ is a } \beta\text{–formula} \\ (\forall \bar{x}\; conv_{\boxed{\forall}}(F'))\wedge\mathbf{1} & \text{if } F \text{ is a } \gamma\text{–formula of the form } \forall \bar{x}\, F' \\ \left(conv_{\boxed{\forall}}(F')\left\{\begin{array}{c} f_1(\bar{y})/x_1 \\ \vdots \\ f_n(\bar{y})/x_n \end{array}\right\}\right)\wedge\mathbf{1} & \text{if } F \text{ is a } \delta\text{–formula of the form } \exists(x_1, \ldots, x_n)\, F', \bar{y} \text{ are the free variables in } F', \text{ and } f_1, \ldots, f_n \text{ are new function symbols} \end{cases}$$

It is assumed that $\bar{x}$ *stands for a finite list of variables* $x_1, \ldots, x_m$. *Skolemization is performed in the "liberalized* δ*–rule" style described in (Hähnle & Schmitt, 1991) on "*$\exists$*"–formulæ; substitutions are written in braces "*$\{\ldots\}$*".*

Figure 1 shows the graph representing a fully expanded tableau for the following problem, taken from Pelletier's problem set (Pelletier, 1986):

Problem 3.3 (Pelletier 30)

Axm_pel30_1:	$\forall x\, (f(x) \vee g(x) \rightarrow \neg h(x))$
Axm_pel30_2:	$\forall x\, ((g(x) \rightarrow \neg i(x)) \rightarrow (f(x) \wedge h(x)))$
Axm_pel30_3:	$\neg\forall x\, i(x)$

The left graph represents the fully expanded tableau for the whole formula, the two graphs for the γ–formulæ are drawn separately to the right of it.

Compiling such a graph for a fully expanded first–order tableau is slightly more complicated than in the propositional case. We will again explain the principle with Prolog as the target language. First, we must choose whether we aim at proving validity or inconsistency. In principle this does not matter, but for historical reasons the latter is usually preferred. We will stick to this, simply because "reversing" things would make them less familiar and therefore less comprehensible.

For each node a clause is generated that succeeds if the paths crossing this node are contradictory. A clause for a binary node succeeds if both clauses for its successor nodes succeed, and a clause for a unary node labeled with a literal L succeeds if either

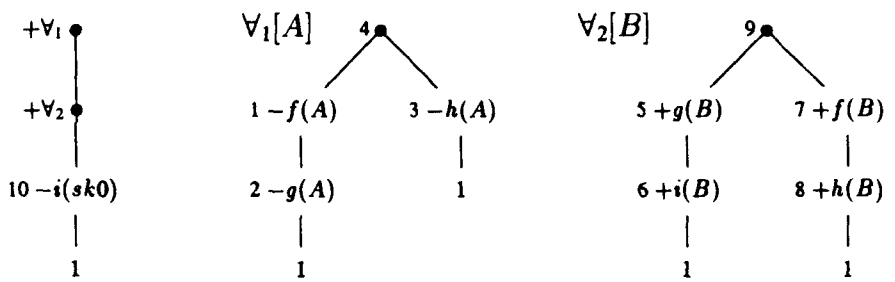


Figure 1: Tableau for Problem 3.3

- there is a substitution σ, such that the current path and L become inconsistent under this substitution, or
- the clause for the successor node succeeds.

If we cross a γ–node for a fully expanded tableau of a γ–formula, we will just note this in the path, but we will not enter the γ–graph, yet. If we arrive with a consistent path at a 1–leaf, one of those γ–tableaux in the path is selected and entered[2].

The basic technical problems that must be solved for the compilation process are:

1. *Variable bindings need to be represented and passed to each clause, since Prolog clauses are – by definition – variable disjoint.*

   This can be solved in the following way: when constructing the graph, all variables that appear are counted. If we have n variables, their binding can be represented by an n–ary Prolog term, each argument holding the binding for the according variable.

2. *It is in general necessary to have more than one instance of an atom (since γ–formulæ may be used multiple), so we will not know the maximal length of a path in advance.*

   For handling this we will represent a path by an open list holding signed instances of atoms[3].

3. *γ–graphs introduce new variable bindings.*

   If variable bindings are handled as pointed out in 1, the Prolog clauses for a γ–graph representing a fully expanded tableau for a γ–formula can be "re–used" arbitrarily often, if a new `binding–term` is created. *The slot(s) holding bin*ding(s) for the quantified variable(s) are simply dropped and left void by inserting

[2]This reflects the usual treatment of γ–formulæ in tableaux: they are not used if the branch can be closed immediately.

[3]There are of course more efficient solutions than this, but we will prefer readability over efficiency, here.

an anonymous Prolog variable. So, it is possible to re–use the tableaux for γ–formulæ without asserting new clauses.

Table 1 shows the complete Prolog program for Problem 3.3. Recall that a clause succeeds if all paths crossing the according node are closed.

Each clause `node/3` "implements" one node. The first parameter is the number of the node in the graph; the second parameter is used to implement a depth–bound on the search that controls the number of applications of γ–formulæ. Paths are represented by an open list holding signed atoms, or, in the clauses "`gamma1`" and "`gamma2`" the name of the clause for the top node of a γ–graph. If the end of a path ("1") is reached without having found a contradiction, `use_gamma/3` selects one of them an calls the entry clause unless a depth bound is reached. `close/2` tries to find a substitution such that a path is closed. Note, that this predicate must enumerate all substitutions during backtracking.

A proof is done by calling the top node (`gamma1`) with the empty path. `proof(2)` succeeds, whereas `proof(1)` fails.

4 Conclusion & Outlook

We have described an approach to tableaux–based theorem proving that works by translating an arbitrary first–order formula into graph representing a fully expanded tableau for it. This graph can then be compiled into a program that performs the proof search when executed. We showed how to do this with Prolog as a target language, although any other general–purpose language can be used.

The principle of compiling formulæ into a general–purpose target language offers several advantages; inter alia,

- it can considerably speed up the proof search,
- it offers a flexible way for embedding deduction into applications, since the compilation can generate stand–alone subroutines that do not require a logical engine to run on.

The presented Prolog code can also be further optimized, especially the first–order version. The propositional version seems to be not very far from an optimum, apart from the fact joining clauses that have only one caller ("unfolding" the Prolog code) might result in more efficient Prolog code.

The method differs from other approaches to deduction by Horn–clause generation (e.g. (Stickel, 1988)), in that

1. the generated program has no direct logical relation to the formula that is to be proven (i.e., the Prolog clauses are not a *logically* equivalent variant of the formula), but that they are *procedurally* equivalent to the search for a model.
2. The method does not require a conjunctive normal form.

```
use_gamma(0,_,_):- !,fail.
use_gamma(Limit,Path,VarBnd) :-
  NewLimit is Limit - 1,
  member(gamma(N),Path),
  node(N,NewLimit,Path,VarBnd).

close(+L1,[H|Path]) :-
  (H = -L2, unify(L1,L2)); close(+L1,Path).
close(-L1,[H|Path]) :-
  (H = +L2, unify(L1,L2)); close(-L1,Path).

node(gamma1,Limit,Path,VarBnd):-
  node(gamma2,Limit,[gamma(4)|Path],VarBnd).

node(gamma2,Limit,Path,VarBnd):-
  node(10,Limit,[gamma(9)|Path],VarBnd).

node(1,Limit,Path,bind(A,B)):-
  (close(-f(A),Path) ;node(2,Limit,[-f(A)|Path],bind(A,B))).
node(2,Limit,Path,bind(A,B)):-
  (close(-g(A),Path) ;use_gamma(Limit,[-g(A)|Path],bind(A,B))).
node(3,Limit,Path,bind(A,B)):-
  (close(-h(A),Path) ;use_gamma(Limit,[-h(A)|Path],bind(A,B))).
node(4,Limit,Path,bind(_,B)):-
  node(1,Limit,Path,bind(A,B)),
  node(3,Limit,Path,bind(A,B)).
node(5,Limit,Path,bind(A,B)):-
  (close(+g(B),Path) ;node(6,Limit,[+g(B)|Path],bind(A,B))).
node(6,Limit,Path,bind(A,B)):-
  (close(+i(B),Path) ;use_gamma(Limit,[+i(B)|Path],bind(A,B))).
node(7,Limit,Path,bind(A,B)):-
  (close(+f(B),Path) ;node(8,Limit,[+f(B)|Path],bind(A,B))).
node(8,Limit,Path,bind(A,B)):-
  (close(+h(B),Path) ;use_gamma(Limit,[+h(B)|Path],bind(A,B))).
node(9,Limit,Path,bind(A,_)):-
  node(5,Limit,Path,bind(A,B)),
  node(7,Limit,Path,bind(A,B)).
node(10,Limit,Path,VarBnd):-
  (close(-i(sk0),Path) ;use_gamma(Limit,[-i(sk0)|Path],VarBnd)).

prove(Limit) :- node(gamma1,Limit,[],_).
```

Table 1: Prolog Program for the Tableau of Figure 1

3. The cost for translating a formula to the proposed representation of a fully expanded tableau, and the compilation of the tableau are both linear (in space an time) w.r.t. the length of the input formula.

One can argue that the proof search in a tableau–based prover usually works by stepwise developing it until each branch in it is closed; therefore, it might happen that branches close with some compound formulæ they are holding. Considering only fully expanded tableaux, as this approach proposes, results in a tableau proof procedure that never closes a branch in this way, but with literals only. From a theoretical point of view this might prevent finding a short proof; in practice, however, it very rarely happens that branches can indeed be closed with compound formulæ. So, it seems justified to neglect this issue.

An experimental implementation of a variant of the method (see (Posegga & Ludäscher, 1992)) has shown the the speedup gained by generating a Prolog program instead of using a "traditional" approach to implementing a tableau prover (without compilation) in Prolog is around a factor of 10. Using Assembler as the target language (which has been implemented for a propositional version) results in a program that runs another 20–30 times faster than the version compiling to Prolog.

References

Melvin C. Fitting. *First–Order Logic and Automated Theorem Proving*. Springer, New York, 1990.

Reiner Hähnle & Peter H. Schmitt. The liberalized δ–rule in free variable semantic tableaux. *to appear*, 1991.

Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191 – 216, 1986.

Joachim Posegga & Bertram Ludäscher. Towards first-order deduction based on shannon graphs. In *Proc. German Workshop on Artificial Intelligence*, LNAI, Bonn, Germany, 1992. Springer.

Joachim Posegga. *First–order Deduction with Shannon Graphs*. PhD thesis, University of Karlsruhe, Karlsruhe, FRG, forthcoming 1993.

Mark E. Stickel. A Prolog Technology Theorem Prover. In E. Lusk & R. Overbeek, editors, *9th International Conference on Automated Deduction*, Argonne, Ill., May 1988. Springer-Verlag.

Short CNF in Finitely–Valued Logics

Reiner Hähnle*

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, 7500 Karlsruhe, Germany
haehnle@ira.uka.de

Abstract. We present a transformation of formulæ from arbitrary finitely–valued logics into a conjunctive normal form based on signed atomic formulæ which can be used to syntactically characterize many–valued validity with a simple resolution rule very much like in classical logic. The transformation is always linear with relation to the size of the input, and we define a generalized concept of polarity in order to remove clauses which are not needed in the proof. The transformation rules are based on the concept of 'sets–as–signs' developed earlier by the author in the context of tableau–based deduction in many–valued logics. We claim that the approach presented here is much more efficient than existing approaches to many–valued resolution.

Introduction

With this paper we make a step toward the efficient mechanization of deduction in many–valued logics. The need for research of that kind is motivated by the recent advent of new applications for many–valued theorem proving in various subfields, for example, in formal hardware verification [5]. Other applications exist in the theory of error–correcting codes or in non–monotonic reasoning.

It is widely acknowledged that the existence of clausal normal forms for a logic can greatly improve efficiency and speed of theorem proving procedures for that logic. Resolution–based theorem provers usually rely on the input being in conjunctive normal form (CNF), but also most other proof procedures that claim high performance, employ CNF transformation as a preprocessing step. If these successful techniques are to be used in non–classical theorem proving, it is likely that some variant of CNF is required for the respective non–classical logics.

There are three main obstacles that have to be overcome when clausal normal forms are to be used in a generalized context:

1. Normal forms can become exponentially long wrt the length of the input when a naïve algorithm is used. This is not so problematic in classical logic where knowledge bases usually consist of conjunctions of relatively short formulæ. In non–classical logics, however, even relatively short formulæ can become quite large during this process already.
2. The normalized input bears no resemblance to the orginal formula. This makes it hard to explain the machine–generated proof to the user.
3. Many non–classical logics may fail to have normal forms, or at least it is non–trivial to find them.

The first two problems can principally be solved by using a *structure preserving clause form translation* (defined in the following section) which has the double advantage of (i) producing normal forms in linear time and space wrt to the input and (ii)

* Research supported by Deutsche Forschungsgemeinschaft (DFG).

establishing a relationship between the clauses of the normal form and the subformulæ of the input formula.

As to the third problem, it is not likely that there is a uniform solution to it due to the diversity of non–classical logics. It has been shown, however, that for certain classes of non–classical logics structure preserving CNF transformations (and corresponding resolution rules) can be devised to give the desired results [7]. These include intuitionistic logic and various modal logics.

The purpose of this note is to define structure preserving clause form translations (together with a suitable definition of clauses and a resolution rule) for arbitrary finitely–valued logics, a domain where, to our best knowledge, no general results exist so far. The normal form computation will be linear wrt to the length of the input and quadratic wrt to the number of truth values in the worst case. The short CNF translation for many–valued logics proposed in the following is centered around a technique that has been developed earlier by the author [3, 4] in connection with *non–clausal* theorem proving with semantic tableaux. The main advantage is a relatively simple resolution procedure for finitely–valued logics which avoids the drawbacks of a non–clausal approach [11], while retaining the main advantages of resolution, notably, strategies for pruning the search space. We treat the propositional case thoroughly and give some hints how to handle the first order case. Due to space restrictions we omit proofs. These may be found in the long version of this paper which is avilable from the author on request.

1 Short Normal Forms in Classical Logic

In the following we denote with $\#(M)$ the cardinality of a set M and with $|s|$ we denote the length of a string s. We use $\lceil x \rceil$ to denote the ceiling function on the rationals. We assume the reader is familiar with the basic notions of computational logic. Throughout the paper we will use a standard syntax for propositional and first–order logic, here and there enriched with some new unary and binary operator symbols. Clauses are considered as finite multisets of literals.

The central idea behind structure preserving clause form translations is to introduce additional atoms which serve as abbreviations for subformulæ of the input. Assume we have a propositional formula ϕ and we need a finite set of clauses X_ϕ such that $\models \phi$ iff $X_\phi \vdash \square$.

Let $SF(\phi)$ denote the set of subformulas of ϕ (note that $\#(SF(\phi)) = |\phi|$) and let $m = \#(SF(\phi))$. We denote with $\bar{L}$ the complement of a literal. Now we introduce a new variable p_i for each $\phi_i \in SF(\phi)$ which is not a literal and consider for each $1 \leq i \leq m$ and $\phi_i = (\phi_j \ \mathsf{op} \ \phi_k)$ the formula

$$p_i \leftrightarrow (p_j \ \mathsf{op} \ p_k) \tag{1}$$

where op is the top–most connective of ϕ_i and p_j, p_k either correspond to ϕ_j, ϕ_k or $\phi_l = p_l$ if ϕ_l is a literal. This process is called *abbreviation*, *definition* or *renaming* by various authors. Let $X_{\phi,i}$ be a CNF representation of (1). The number of clauses in $X_{\phi,i}$ is bound by a constant depending on the type of connectives present and is at most 4; each clause contains at most 3 literals. Now we can define

$$X_\phi = \left(\bigcup_i X_{\phi,i} \right) \cup \{\bar{p}_1\} \tag{2}$$

where p_1 is the definition of ϕ. It is fairly easy to see that X_ϕ has indeed the desired properties, in particular, X_ϕ contains at most $12m + 1$ literals.

In the first–order case the p_i are atomic formulæ with an appropriate arity.

Example 1. Consider the propositional tautology $p \supset (q \supset p)$. We introduce the following renamings: $p_1 \rightarrowtail p \supset p_2$, $p_2 \rightarrowtail q \supset p$. So the formula is a tautology iff the following set of clauses is unsatisfiable:

$$\begin{array}{llll} 1. & \sim p_1 & & \\ 2. & \sim p_1 \vee p_2 \vee \sim p & & \\ 3. & p_1 \vee p & & \\ 4. & p_1 \vee \sim p_2 & & \end{array} \qquad \begin{array}{llll} 5. & \sim p_2 \vee p \vee \sim q & & \\ 6. & p_2 \vee & & q \\ 7. & p_2 \vee \sim p & & \end{array}$$

A refutation of this clause set is as follows:

$$\begin{array}{llll} 8. & [1,3] & p & \\ 9. & [1,4] & \sim p_2 & \end{array} \qquad \begin{array}{llll} 10. & [7,8] & p_2 & \\ 11. & [9,10] & \square & \end{array}$$

Note that 2., 5. (each corresponding to one half of an equivalence) and 6. were not needed. Obviously, our CNF contains redundant clauses.

There is a rather obvious improvement of the procedure, if one observes that instead of logical equivalence in (1), depending on the polarity (cf. [10]) of p_i in the original formula (ie in $\sim \phi$), only one direction of the implication is needed in order to characterize satisfiability. Therefore, instead of (1) we write

$$\begin{array}{l} p_i \supset (p_j \ \mathsf{op} \ p_k) \text{ if } p_i \text{ occurs positively in } \sim \phi \\ (p_j \ \mathsf{op} \ p_k) \supset p_i \text{ if } p_i \text{ occurs negatively in } \sim \phi \\ p_i \rightarrowtail (p_j \ \mathsf{op} \ p_k) \text{ if } p_i \text{ occurs positively and negatively in } \sim \phi \end{array} \tag{3}$$

If we apply this optimization to the previous example, clauses 2. and 5. are not generated.

Further improvements are possible if not all subformulas are being renamed, for instance, conjunctions need not to be renamed. An optimal result (the clause set $\{p, \sim p, q\}$) would have been obtained using a *top–down renaming* algorithm [2] which takes this into account. In [2] one may also find additional references regarding structure preserving clause form for classical logic.

2 Many–Valued Logic

Definition 1 Syntax, Truth Values. Let $\mathbf{L}$ be a propositional language with propositional variables $\mathbf{L_0}$ and connectives $\mathbf{F}$. Let $N = \left\{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\right\}$ be the set of **truth values** and let $n = \#(N)$.

Definition 2 Semantics, Many–Valued Logic. Connectives $F \in \mathbf{F}$ are interpreted as functions with finite range and domain, in other words, if k is the arity of F we associate a function $f : N^k \rightarrow N$ with F which we call the **interpretation** of F. Let $\mathbf{f}$ be the family of functions over N associated with connectives in $\mathbf{F}$. Then we call $\mathbf{f}$ n–valued matrix for $\mathbf{L}$ and the triple $\langle \mathbf{L}, \mathbf{f}, N \rangle$ **n–valued propositional logic.**

Definition 3 Valuation. Let $\mathcal{L} = \langle \mathbf{L}, \mathbf{f}, N \rangle$ be a n–valued propositional logic. A **valuation** for $\mathcal{L}$ is a function $v : \mathbf{L_0} \rightarrow N$. As usual, v can be uniquely extended to a homomorphism from $\mathbf{L}$ to N via

$$v(F(\phi_1, \ldots, \phi_k)) = f(v(\phi_1), \ldots, v(\phi_k))$$

where f is the interpretation of F.

Definition 4 S**–Satisfiable,** S**–Tautology.** For $S \subseteq N$ and a n–valued propositional logic $\mathcal{L}$ call a formula $\phi \in \mathbf{L}$ S**–satisfiable** iff there is a valuation such that $v(\phi) \in S$. Call ϕ a S**–tautology** iff $v(\phi) \in S$ for all valuations.

For some examples we refer the reader to the following section. Our task is now to find

1. a language of clauses C;
2. a structure preserving linear translation tr from $\mathbf{L} \times 2^N$ into 2^C;
3. a resolution rule on C, i.e. a decidable relation $R \subseteq C^{k+1}$ for some k.

such that $\mathrm{tr}(\phi, S) \vdash \square$ iff ϕ is a S–tautology (where $\vdash$ is the reflexive and transitive closure of R).

3 A Structure Preserving Normal Form Translation for Many–Valued Logics

In [3, 4] the author introduced semantic tableau systems that can be used to implement a generic theorem prover which performs efficiently in a variety of finitely–valued logics. The key idea was to enhance the formula language in such a way that the still to be considered valuations at each step of the proof can efficiently be kept track of. The technical device was the use of truth value sets as signs or prefixes in front of the formulæ. We define the set of signed formulæ $\mathbf{L}^* = \{S : \phi | S \subseteq N, \phi \in \mathbf{L}\}$ with the intended meaning

$$S : \phi \text{ is satisfiable iff } v(\phi) \in S \text{ for some } v.$$

Example 2. A sound and complete rule with premise $\{\frac{1}{2}\} : \phi \vee \psi$, where $\vee$ is three–valued strong Kleene disjunction (which is defined $v(\phi \vee \psi) = \max(v(\phi), v(\psi))$) is

$$\frac{\{\frac{1}{2}\} : \phi \vee \psi}{\begin{array}{l|l} \{0, \frac{1}{2}\} : \phi & \{\frac{1}{2}\} : \phi \\ \{\frac{1}{2}\} : \psi & \{0, \frac{1}{2}\} : \psi \end{array}}$$

One way to visualize rules with a premise $S : (\phi \ \mathsf{op} \ \psi)$ uses coverings of those entries in the truth table of op that are members of the set S. Each of the rule extensions corresponds to a partial covering of these entries. The union of all coverings corresponds to the collection of extensions that make up the conclusion of a rule. In Example 2 above, the left extension covers the area indicated in the following diagram on the left, while the right extension covers the area shown in the diagram on right.

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

Now, tableau rules and tableaux correspond to DNF formulæ, while we are interested in CNF formulæ. What we need, therefore, are *inverse* tableau rules where the extensions are conjunctively connected and the extensions themselves are clauses over signed atoms. Consequently, we define the language of clauses C to be the clauses over $\mathbf{L}^*_{\mathbf{0}}$. The $\vee$ in C will be interpreted classically, i.e. two–valued (like the implicit disjunctions of tableau branches in many–valued tableaux which are also interpreted classically):

Definition 5 Satisfiability on C. An atomic signed formula $S : p$ is satisfied by v iff $v(p) \in S$. Let $D \in C$ and $D = S_1 : p_1 \vee \cdots \vee S_k : p_k$. D is satisfied by v iff v satisfies at least one $S_i : p_i$ in D. A clause set $X \subseteq C$ is satisfied by v iff v satisfies simultaneously each member of X. We write $v \vDash X$ for this fact. X is **satisfiable** iff $v \vDash X$ for some v.

How can we compute inverse tableau rules? We can still use the technique with coverings, however, we must turn things around. Each extension (or clause) corresponds to a covering that contains at least the entries occurring in S. The *intersection* of all coverings must contain exactly the entries occurring in S.

Example 3. The inverse tableau rule with the premise of Example 2 is:

$$\frac{\{\frac{1}{2}\} : \phi \vee \psi}{\{0, \frac{1}{2}\} : \phi \;\|\; \{0, \frac{1}{2}\} : \psi \;\|\; \begin{matrix}\{\frac{1}{2}\} : \phi \\ \{\frac{1}{2}\} : \psi\end{matrix}}$$

The extensions correspond to the following coverings:

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

The conclusion of the rule corresponds to the following set of C–clauses:

$$\{\{0, \tfrac{1}{2}\} : \phi,\ \{0, \tfrac{1}{2}\} : \psi,\ \{\tfrac{1}{2}\} : \phi \vee \{\tfrac{1}{2}\} : \psi\}$$

We draw inverse tableau rules with double vertical bars to distinguish them from the ordinary rules.

The next step is to express logical equivalence within this framework. Consider the following definition of (strong) **many–valued equivalence**:

$$v(\phi \leftrightharpoons \psi) = \begin{cases} 1 & v(\phi) = v(\psi) \\ 0 & \text{otherwise} \end{cases}$$

Let us give a formulation with C–clauses of $\leftrightharpoons$. It will be convenient to use the following abbreviations for signs:

Definition 6 Signs. Let $j \in N$ be arbitrary.

$$\boxed{\geq j} := [j, 1] \cap N \qquad \boxed{> j} := (j, 1] \cap N$$
$$\boxed{\leq j} := [0, j] \cap N \qquad \boxed{< j} := [0, j) \cap N$$

Now consider the $(2n - 2)$ C–clauses of the following form:

$$\begin{matrix}\boxed{\leq j} : p \vee \boxed{> j} : q \\ \boxed{> j} : p \vee \boxed{\leq j} : q\end{matrix} \quad \text{where } j < 1 \tag{4}$$

Let us denote this clause set with $X_{\leftrightharpoons}$; it is easy to prove that $p \leftrightharpoons q$ is $\{1\}$–satisfiable iff $X_{\leftrightharpoons}$ is satisfiable and $p \leftrightharpoons q$ is $\{0\}$–satisfiable iff $X_{\leftrightharpoons}$ is unsatisfiable.

We have now all prerequisites for mimicking the structure preserving clause form translation described in (1) and (2) in the many–valued case. We can apply the very same procedure for computing a CNF over C–clauses for some formula ϕ as in the classical case. For each formula of the form (1) we simply expand the signed formula

$$\{1\} : (p_i \leftarrow (p_j \ \mathsf{op} \ p_k)) \tag{5}$$

using the inverse tableau rules for $\leftarrow$ and op (recall that the conclusion corresponds to a set of C–clauses, namely to $X_{_}$). This process yields a set $X_{\phi,i}$ for each non–atomic subformula ϕ_i of ϕ. To establish S–satisfiability of ϕ for some $S \subseteq N$ it is sufficient to show that

$$X_\phi := \left(\bigcup_i X_{\phi,i} \right) \cup \{S^c : p_1\} \tag{6}$$

is unsatisfiable (where p_1 is the renaming of ϕ). Since the branching factor of each inverse tableau rule is at most n we have:

Corollary 7. *In every n–valued logic $\mathcal{L}$ for every $S : \phi \in L^*$ there is a C–CNF representation X_ϕ of ϕ of length $\mathcal{O}(n^2|\phi|)$ such that $S : \phi$ is valid iff $X_\phi \vdash \square$.*

Let us illustrate the method with an example.

Example 4. Consider three–valued strong Kleene logic with an extra negation '$\sim$' (called $\widetilde{\mathbf{SKL}}$). For convenience we repeat the semantic definitions (which are valid for any number of truth values): $v(\neg\phi) = 1 - v(\phi)$, $v(\sim \phi) = \ulcorner 1 - v(\phi) \urcorner$, $v(\phi \wedge \psi) = \min(v(\phi), v(\psi))$, $v(\phi \vee \psi) = \max(v(\phi), v(\psi))$, $v(\phi \supset \psi) = \max(1 - v(\phi), v(\psi))$.

To establish that $\neg p \supset (\sim p \wedge \neg p)$ is a tautology (in $\widetilde{\mathbf{SKL}}$ both $\frac{1}{2}$ and 1 are designated truth values, i.e. support validity) we need to show that it is a $\{\frac{1}{2}, 1\}$–tautology which is the case iff the C–CNF of $\{0\} : \neg p \supset (\sim p \wedge \neg p)$ is unsatisfiable. To simplify things a bit we introduce no new variables for negated atoms. Thus we have

$$\begin{array}{l} \{0\} : q \\ \{1\} : (q \leftarrow (\neg p \supset r)) \\ \{1\} : (r \leftarrow (\sim p \wedge \neg p)) \end{array} \tag{7}$$

We begin to expand the second formula (cf. (4)):

$$\{1\} : (q \leftarrow (\neg p \supset r))$$

$$\begin{array}{c|c|c|c} \{\frac{1}{2}, 1\} : q & \{0\} : q & \{0, \frac{1}{2}\} : q & \{1\} : q \\ \{0\} : \neg p \supset r & \{\frac{1}{2}, 1\} : \neg p \supset r & \{1\} : \neg p \supset r & \{0, \frac{1}{2}\} : \neg p \supset r \end{array}$$

The formulæ containing an implication have to be expanded further in order to yield the clause set X_q. Similarly, we compute the set X_r. Together with the first clause in (7) we arrive at the following set of signed clauses which characterizes the original problem:

1. $\{0\} : q$
2. $\{\frac{1}{2}, 1\} : q \vee \{0\} : r$
3. $\{\frac{1}{2}, 1\} : q \vee \{0\} : p$
4. $\{0\} : q \vee \{\frac{1}{2}, 1\} : p \vee \{\frac{1}{2}, 1\} : r$
5. $\{0, \frac{1}{2}\} : q \vee \{1\} : p \vee \{1\} : r$
6. $\{1\} : q \vee \{0, \frac{1}{2}\} : p$
7. $\{1\} : q \vee \{0, \frac{1}{2}\} : r$
8. $\{1\} : p \vee \{\frac{1}{2}, 1\} : r$
9. $\{0, \frac{1}{2}\} : p \vee \{0\} : r$
10. $\{0\} : p \vee \{0, \frac{1}{2}\} : r$
11. $\{\frac{1}{2}, 1\} : p \vee \{1\} : r$

4 Signed Resolution

We still have to provide a resolution rule for C–clauses. The following rule is one of several possibilities and very close to standard binary resolution:

$$\frac{S_1 : p \vee D_1 \quad \cdots \quad S_m : p \vee D_m}{D_1 \vee \cdots \vee D_m} \qquad \text{if } S_1 \cap \cdots \cap S_m = \emptyset \tag{8}$$

For completeness we need a factoring rule due to similar reasons as in the classical case.

$$\frac{S_1 : p \vee \cdots \vee S_m : p \vee D}{(S_1 \cup \cdots \cup S_m) : p \vee D} \tag{9}$$

Soundness of rules (8) and (9) is straightforward to show. Completeness, too, is not hard to prove with a semantic tree argument that is readily generalized to more than two truth values by allowing n–ary semantic trees. In [1] a similar result is proved with that method and can readily be adapted to the present case.

Example 5. We continue Example 4 and show that the set of C–clauses generated there is unsatisfiable:

12.	[1,2]	$\{0\} : r$	14.	[8,12]	$\{1\} : p$
13.	[1,3]	$\{0\} : p$	15.	[13,14]	$\square$

Note that only the input clauses 1., 2., 3. and 8. have actually been used in the derivation. We will come back to this issue in the next section.

5 Improvements

We present a simplification which parallels (3). We have already seen in Example 5 that most of the clauses were not redundant. We have to define a generalized notion of polarity in the presence of more than two truth values.

Definition 8 Many–Valued Polarity. Let $S : \phi \in \mathbf{L}^*$ and let $\mathbf{T}$ be a fully expanded inverted tableau for $S : \phi$. For each subformula ψ of ϕ, if the occurrences[2] of ψ in $\mathbf{T}$ are $S_1 : \psi, \ldots, S_m : \psi$, we say that ψ occurs with **polarity** $R = \langle S_1, \ldots, S_m \rangle$ in ϕ. We abbreviate this fact with $R : \psi \prec S : \phi$.

Note that by definition of inverted tableau rules $\emptyset \not\subseteq S_i \not\subseteq N$ holds for each S_i in R. For each polarity R we define a binary connective $\Rightarrow_R$ by

$$v(\phi \Rightarrow_R \psi) = \begin{cases} 0 & \text{if } v(\psi) \notin S_i,\ v(\phi) \in S_i,\ S_i \text{ occurs in } R \\ 1 & \text{otherwise} \end{cases}$$

We observe that in two–valued logic $\Rightarrow_{\{\{1\}\}}$ is the same connective as $\supset$ and $\Rightarrow_{\{\{0\}\}}$ is the same connective as $\subset$. Moreover, $\Rightarrow_{\{\{0\},\{1\}\}}$ is the same connective as $\leftrightarrow$. Now we replace (5) by

$$\{1\} : (p_i \Rightarrow_R (p_j \ \mathsf{op} \ p_k)), \text{ if } R : (p_j \ \mathsf{op} \ p_k) \prec S : \phi. \tag{10}$$

In two–valued logic we can get rid of the signs simply by writing everywhere p for $\{1\} : p$ and $\sim p$ for $\{0\} : p$. Together with the observation above (10) collapses into (3) for two–valued logic, if we associate positive polarity with $\langle\{1\}\rangle$, negative polarity with $\langle\{0\}\rangle$ and both polarities with $\langle\{0\},\{1\}\rangle$.

[2] Be aware that identical strings can be different subformulas, such as p in $p \supset p$. On the other hand, the same subformula can occur multiply in the tableau, since they are copied in some rules, such as ϕ in Example 2.

Example 6. Let us apply Definition 8 to (7) from Example 4. Obviously, $\neg p \supset (\sim p \wedge \neg p)$ occurs with polarity $\langle\{0\}\rangle$ in $\{0\} : \neg p \supset (\sim p \wedge \neg p)$. To see the polarity of $\sim p \wedge \neg p$ we begin to compute the inverse tableau for $\{0\} : \neg p \supset (\sim p \wedge \neg p)$.

$$\frac{\{0\} : \neg p \supset (\sim p \wedge \neg p)}{\{1\} : \neg p \| \{0\} : \sim p \wedge \neg p}$$

We see that the polarity of $\sim p \wedge \neg p$ is $\langle\{0\}\rangle$, too. Therefore, we substitute both occurrences of — in (7) by $\Rightarrow_{\langle\{0\}\rangle}$:

$$\begin{aligned} &\{0\} : q \\ &\{1\} : (q \Rightarrow_{\langle\{0\}\rangle} (\neg p \supset r)) \\ &\{1\} : (r \Rightarrow_{\langle\{0\}\rangle} (\sim p \wedge \neg p)) \end{aligned} \tag{11}$$

As an easy exercise the reader should verify that the clause set corresponding to $\{1\} : (q \Rightarrow_{\langle\{0\}\rangle} (\neg p \supset r))$ is $\{\{\frac{1}{2}, 1\} : q \vee \{0\} : r, \{\frac{1}{2}, 1\} : q \vee \{0\} : p\}$ and the clause set corresponding to $(r \Rightarrow_{\langle\{0\}\rangle} (\sim p \wedge \neg p))$ is $\{\{\frac{1}{2}, 1\} : r \vee \{1\} : p\}$. But these are exactly clauses 2., 3 and 8. from the old clause set (cf. Example 5) and they were exactly the ones used in the refutation. Hence we succeeded in eliminating all clauses that were redundant in Example 5.

Theorem 9. *Let X^1_ϕ be a set of C–clauses corresponding to some $\phi \in L$ computed according to (5) and let X^2_ϕ be a set of C–clauses computed according to (10). Then $X^1_\phi \vdash \square$ iff $X^2_\phi \vdash \square$.*

Unfortunately, in the worst case the number of generated clauses can still be quadratic wrt the number of truth values (use the same example as before). If, however, the maximal number of occurrences of either ϕ or ψ with different signs in the conclusion of each rule with premise $S : \phi$ **op** ψ for all signs S and connectives **op** in a logic is $k \leq n$ we can replace $\mathcal{O}(n^2|\phi|)$ by $\leq k^2|\phi| + l$ for some l in the corollary above. In classical logic we have $k = 1$ if no equivalences are present and $k = n = 2$ otherwise. See [4] for a class of many–valued logics where $k = 1$.

We suspect that much of the work in [2] can be generalized to the many–valued case, in particular, it should be possible to prove the optimality of certain many–valued CNF translations under suitable restrictions on the connectives.

Other possible improvements regard the resolution rule. We state some well-known strategies from classical resolution in our many–valued setting. This strengthens our claim that signed resolution is a natural extension of two–valued resolution.

Definition 10 Subsumption. Let D, E be two C–clauses. We say that D **is subsumed** by E iff for each literal $S_1 : p$ in E there is a literal $S_2 : p$ in D such that $S_1 \subseteq S_2$.

Having this definition at hand, we can formulate subsumption strategies as in the classical case.

Our final point in this section is that among other strategies the set–of–support strategy, as well as a pure rule and deletion of tautologies (clauses containing a literal of the form $N : p$) may be formulated and proved as complete based on our notion of satisfiability just as in the two–valued case.

6 Related Work

We have seen that it is possible to compute short ((~)CNF for finitely–valued logics in a quite efficient way using truth value sets as signs and inverse tableau rules. The resulting set of signed clauses is "flattened out" and provides no proof–theoretic insight. In [7] (for modal and intuitionistic logic) and [8] (for Łukasiewicz logic) a different view is taken: there, a logic is characterized by clause sets whose syntax does not involve signs, instead, certain additional logical connectives like necessity $\Box$ [7] or truncated sum $\dot{\vee}$ [8] are used. The problem with this approach is that it cannot be done schematically—each new logic requires new ideas. Also a new completeness proof of the associated resolution rule has to be carried out each time.

In [1] a similar translation method and resolution rule as developed in Sections 3 and 4 can be found. It is stated in somewhat different terms and, what is more important, uses only single truth values as signs. Also the translation is not linear and does not employ polarity. Thus we conjecture that our own approach is more amenable to implementation. In [9] another variant of signed resolution is investigated.

In [11] a kind of many–valued polarity is introduced for the purpose of pruning the enormous search space in non–clausal resolution. O'Hearn & Stachniak's polarity notion is defined on unsigned formulæ and close to the original definition of Murray [10]. We do not see any resemblance to the polarity notion developed in this paper.

In [6] the notion of p–resolution is defined in the context of automated reasoning in paraconsistent logics. These are truth value lattice–based logics and resolving between literals involves not only mere set intersection and union as in rules (8,9), but meet and join operations on the truth value lattice. On the other hand, in the case of linear orders of truth values and restriction to signs $\boxed{\geq i}$, $\boxed{\leq i}$ p–resolution essentially coincides with rules (8,9). This fact suggests that our method can be expanded to the treatment of *non–clausal paraconsistent logics* ([6] assume to have the input in signed CNF).

7 First–Order Logic

We consider many–valued versions[3] of $\forall$, $\exists$ which have the simplified Skolem conditions given in Table 1. Other signs than $\boxed{\geq i}$, $\boxed{\leq i}$ are handled by *splitting*:

$$\frac{S : \phi}{\begin{array}{l} S_1 : \phi \\ \vdots \\ S_k : \phi \end{array}} \qquad \text{where } S = S_1 \cup \cdots \cup S_k.$$

$$\frac{\{i\} : \phi}{\boxed{\geq i} : \phi \;\Big\|\; \boxed{\leq i} : \phi}$$

Adding these expansion rules extends the translation method given in Section 3 to first–order formulæ.

Soundness and completeness proofs can easily be obtained by combining the usual results on Skolemization with the results of [4].

[3] Defined as $v_\beta((\forall y)\phi) = \min\{v_{\beta^u_y}(\phi) | u \in U\}$, $v_\beta((\exists y)\phi) = \max\{v_{\beta^u_y}(\phi) | u \in U\}$, where min, max are interpreted naturally on N and U is the underlying universe.

Table 1. Simplified Skolem rules for quantified formulæ.

$$\frac{\boxed{\underline{\ }i} : (\forall x)\phi(x)}{\boxed{\geq i} : \phi(x)} \qquad \frac{\boxed{\leq i} : (\forall x)\phi(x)}{\boxed{\leq i} : \phi(f(x_1, \ldots, x_n))}$$

$$\frac{\boxed{\leq i} : (\exists x)\phi(x)}{\boxed{\leq i} : \phi(x)} \qquad \frac{\boxed{\geq i} : (\exists x)\phi(x)}{\boxed{\geq i} : \phi(f(x_1, \ldots, x_n))}$$

(f is a new function symbol and $x_1, \ldots, x_n$ are the variables which occur freely in the premise of the rule.)

8 Conclusion and Future Work

We presented the first steps towards an efficient resolution system for many–valued logics by specifying a way to produce sets of clauses of feasible size which characterize the original problem. The use of short normal form algorithms is not very widespread in classical logic, since they are not really necessary there for most problems. For many–valued logics, however, their use becomes essential, in particular when applications like hardware verification is aimed at, where large formulæ are to be expected.

In our CNF algorithm we defined a generalized notion of polarity which allows to remove redundant clauses. This is, however, only the first part. The next step would be to translate the top–down renamings of [2] in a many–valued setting. For the resolution procedure we sketched some familiar improvements like subsumption in the many–valued version. Further work could include the investigation of successful strategies such as hyper–resolution and ordered resolution; see [1] for first steps in that direction.

References

1. M. Baaz and C. G. Fermüller. Resolution for many–valued logics. In *Proc. LPAR'92*, pp. 107 – 118. Springer, LNAI 624, 1992.
2. T. Boy de la Tour. Minimizing the number of clauses by renaming. In *Proc. 10^{th} CADE, Kaiserslautern*, pp. 558 – 572. Springer, Heidelberg, July 1990.
3. R. Hähnle. Towards an efficient tableau proof procedure for multiple–valued logics. In *Proc. Workshop Computer Science Logic, Heidelberg*, pp. 248 – 260. Springer, LNCS 533, 1990.
4. R. Hähnle. Uniform notation of tableaux rules for multiple–valued logics. In *Proc. 20^{th} ISMVL, Victoria*, pp. 238 – 245. IEEE Press, 1991.
5. R. Hähnle and W. Kernig. Verification of switch–level circuits with multiple–valued logics. *To appear*, 1993.
6. J. J. Lu, L. J. Henschen, V.S. Subrahmanian, and N. C. A. da Costa. Reasoning in paraconsistent logics. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pp. 181 – 210. Kluwer, 1991.
7. G. Mints. Gentzen–type systems and resolution rules, part 1: Propositional logic. In *Proc. COLOG–88, Tallin*, pp. 198 – 231. Springer, LNCS 417, 1990.
8. D. Mundici. Normal forms in infinite–valued logic: The case of one variable. In *Proc. Workshop Computer Science Logic 91, Berne*. Springer, LNCS, 1991.
9. N. Murray and E. Rosenthal. Resolution and path–dissolution in multiple–valued logics. In *Proc. ISMIS, Charlotte*, 1991.
10. N. V. Murray. Completely non-clausal theorem proving. *AI*, 18:67 – 85, 1982.
11. P. O'Hearn and Z. Stachniak. A resolution framework for finitely–valued first–order logics. *Journal of Symbolic Computing*, 13:235 – 254, 1992.

Defining Variants of Default Logic: a Modal Approach

Laura Giordano
Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 TORINO
E-mail: laura@di.unito.it

Abstract. Recently some variants of Reiter's default logic have been proposed. These variants have been defined by altering the definition of default extension and, sometimes, also the definition of default theory. Recently a uniform semantic framework has been introduced by Besnard and Schaub, in which the semantics of the various default logics is given in terms of Kripke structures.
In this paper a uniform syntactic characterization for these different default logics is presented. First, a modal default logic, called K-default logic, is introduced. This logic is defined similarly to Reiter's default logic but it is based on an underlying modal logic (instead of classical logic). We show how the different variants of default logic, like Schaub's Constrained Default Logic, Brewka's CDL and also Lukaszewicz' variant, can be reconstructed within KDL: for each variant a different modal translation of default rules is proposed. In this way, the differences among the variants are made explicit on a syntactic ground.

1. Introduction

Recently some variants of Reiter's default logic (DL) [Reiter80] have been proposed [Lucaszewicz88, Brewka91, Delgrande&Jackson91, Schaub91b], to cope with some unintuitive results classical default logic may lead to. These variants have been defined by altering the definition of default extension and, sometimes, also the definition of default theory, as in Brewka's Cumulative Default Logic (CDL). For these variants a uniform semantic framework has been introduced by Besnard and Schaub [Besnard&Schaub92] in terms of Kripke structures.

In this paper the problem of defining a uniform syntactic characterization for these different default logics is addressed. To this purpose a modal default logic, called K-default logic (KDL), is introduced. KDL is defined like Reiter's default logic with the difference that the underlying monotonic logic is not classical logic but the modal logic K. The idea is that in such a modal default logic the presence of modal operators allows to capture the differences among the variants on a syntactic level, in the way default rules are stated. While Reiter's default logic does not commit to assumptions and in the definition of DL extension no track is kept of justifications of applied defaults, in the variants mentioned above these and other additional information are recorded in constructing extensions. Accordingly, the applicability condition for

default rules is strengthened. In a modal default logic we can explicitly keep track of these additional information by making use of modal operators.

While in Section 2 KDL is defined and some of its properties are stated, in Section 3 and 4 it is shown how Schaub's Constrained default logic, Brewka's CDL and Lukaszewicz' variant, can be reconstructed within KDL: for each variant a translation from default theories to K-default theories is given. In particular, the different variants require a different modal translation of default rules. In this way, the differences among the variants are made explicit in syntactic terms.

As observed by Besnard and Schaub the notion of commitment to assumptions [Poole89] can be given a very natural interpretation in a modal setting. Indeed, the notion of commitment has also a rather simple syntactic interpretation in KDL. In Section 5 we show how this suggests an alternative variant of default logic, which is quite similar to Brewka's CLD, is cumulative but is not semimonotonic, and embodies this notion of commitment to assumptions.

2. KDL: a modal default logic

In this section we define K-default logic (KDL), a default logic whose underlying monotonic logic is the modal logic K. In K-default logic modal operators are allowed both in the set W of formulas and in the prerequisite, justification and consequent of default rules.

For simplicity, in the following we will restrict our concern to the propositional case. Let L be a propositional (non modal) language and L^* the corresponding modal language in which, as usual, $\Box$ and $\Diamond$ are the universal and existential modal operators. We recall Kripke semantics for propositional K modal logic [Hughes&Cresswell68, Bowen79]. We will refer to the language containing the logical connectives $\wedge$ and $\supset$.

Let A_L be the set of all propositional symbols in L^*. A ***K-interpretation*** for L^* is a triple M=<W,R,e,w>, where W is a set of worlds, R is a binary relation on W (the accessibility relation), e is a valuation function $e : W \rightarrow \mathcal{P}(A_L)$ and w is a distinguished world of W. We define the ***satisfiability*** of a closed formula α of L^* at a world $w \in W$ in an K-interpretation M ($M,w \models_K \alpha$) as follows:

- $M,w \models_K p$ iff $p \in e(w)$ (if p is in A_L)
- $M,w \models_K \alpha \wedge \beta$ iff $M,w \models_K \alpha$ and $M,w \models_K \beta$
- $M,w \models_K \alpha \supset \beta$ iff $M,w \not\models_K \alpha$ or $M,w \models_K \beta$
- $M,w \models_K \Diamond \alpha$ iff there is a world $w' \in W$ such that w R w' and $M,w' \models_K \alpha$
- $M,w \models_K \Box \alpha$ iff for all worlds $w' \in W$ such that w R w', $M,w' \models_K \alpha$.

A formula α is ***true*** in a K-interpretation M= <W,R,e,w> (written $M \models_K \alpha$) iff $M,w \models_K \alpha$. We say that α is a ***K-valid*** formula ($\models_K \alpha$) iff it is *true* in every K-interpretation for L^*, i.e., for every K-interpretation M=<W,R,e,w>, $M,w \models_K \alpha$. A

formula α is ***K-consistent*** if there is a K-interpretation M= <W,R,e,w> such that $M,w \models_K \alpha$. In the following we will denote by $Th_K(A)$, the set of logical consequences of A in the logic K, that is:

$$Th_K(A) = \{ \alpha : \models_K A \supset \alpha \}.$$

Let us now define the notions of K-default theory and KDL extension.

Definition 1. A ***K-default theory*** is a pair (D,W) where W is a set of formulas of the modal language L^* and D is a set of default rules of the form A:B/C, where A, B and C are formulas of L^*.

Definition. A ***KDL extension*** of a K-default theory (D,W) is a fixpoint of the operator Γ which, given the set of L^* formulas S, produces the smallest set of L^* formulas S' such that:

(1) $W \subseteq S'$,

(2) S' is closed wrt logical consequence in K, i.e.,$Th_K(S')=S'$,

(3) if $(A:B/C) \in D$, $A \in S'$ and $\neg B \notin S$, then $C \in S'$.

Note that this definition of KDL extension is exactly the same as Reiter's definition apart from the fact that S' is required to be closed wrt logical consequence in K and not in classical logic as usual. Hence, most of the properties of DL hold also for KDL. In particular, as usual, an equivalent quasi-inductive definition of KDL extension can be given.

Definition. Let (D,W) be a K-default theory. Define $E_0=W$ and, for all i>0,

$$E_{i+1}= Th_K(E_i) \cup \{C \mid (A:B/C) \in D, \ A \in E_i \ \text{ and } \ \neg B \notin E\}.$$

E is a KDL extension of (D,W) iff $E= \cup_{i=0,\omega} E_i$.

Let us now consider some examples to see how the presence of modal operators in the language provides more flexibility in writing default rules.

Example 1:

$$D=\{:B/C\ ,\ :\neg B/\ D\},\ W=\{\}.$$

(D,W) has a unique (Reiter) default extension E= Th({C,D}), which is obtained by applying both the defaults, though their justifications are mutually inconsistent. Similarly, $E'=Th_K(\{C,D\})$ is the unique KDL extension of (D,W). If we want to require joint consistency of justifications, we can rewrite the default theory above as follows:

$$D_K=\{:\Diamond B/ (C \wedge \Box C \wedge \Box B)\ ,\ :\Diamond \neg B /(D \wedge \Box D \wedge \Box \neg B)\ \},\ W_K=\{\}.$$

The default theory (D_K,W_K) has two KDL extensions $E_1=Th_K(\{C, \Box C, \Box B\})$, obtained by applying the first default, and $E_2=Th_K(\{D, \Box D, \Box \neg B\})$, obtained by applying the second one. In extension E_1 the second default is not applicable since the condition $\neg \Diamond \neg B \notin E_1$ does not hold (in fact $\Box B \in E_1$).

In the theory (D_K,W_K) the modal formulas $\Box B$ and $\Box \neg B$ introduced in the consequent of default rules are used to commit to the assumptions, i.e. to record the justifications of default rules when they are applied. Notice that, since B is an

assumption underlying extension E_1, $\Box B$ is in E_1; however, B is not true in E_1. Hence, modal operators can be used to distinguish among the beliefs that are true in an extension (represented by non-modal formulas like C in extension E_1) and the assumptions supporting those beliefs (represented by modal formulas like $\Box B$ in extension E_1).

Of course, when a K-default theory does not contain modal operators (neither in D nor in W) its KDL extensions are in a one to one correspondence with DL extensions. Hence, as in Reiter's default logic, also in KDL existence of extensions is not guaranteed. For instance the default theory (D,W), with D={:A/¬ A} and W={}, has no KDL extensions. Moreover, the following propositions hold.

Proposition 1. A K-default theory (D,W) has a K-inconsistent KDL extension iff W is K-inconsistent.

Proposition 2. If a K-default theory (D,W) has a K-inconsistent KDL extension then this is its only KDL extension.

In the next sections it will be shown how constrained default logic [Schaub91b], CDL [Brewka91], and Lukaszewicz' version of default logic [Lucaszewicz88] can all be mapped to K-default logic, while preserving their extensions. The idea underlying these mappings is that of using modal operators to record the additional information (justifications and consequents) these variants keep track of.

3. Mapping constrained default logic to KDL

Starting from the observation that Reiter's default logic lacks desirable features like "cumulativity" and "commitment to assumptions", some variants of DL have been proposed. Brewka has defined a Cumulative Default Logic (CDL) [Brewka91] in which *assertions*, i.e. formulas labelled with a *support* are introduced. In the support of a formula, the justifications and consequents of defaults used to derive the formula are recorded. In this way cumulativity is obtained and also the problem of mutually inconsistent justifications is solved.

Other cumulative variants of default logic, Constrained Default Logic [Schaub91b] and J-Default Logic [Delgrande&Jackson91], turn out to be equivalent. These variants do not make use of assertions as CDL, but (in the style of Lukaszewicz) they define extensions as pairs of sets of formulas (E,C), where C is the context supporting the beliefs in E. In spite of this difference, these logics are very similar to CDL and an equivalence result between Constrained Default Logic and CDL is given in [Schaub92].

In this section we will define an interpretation of Constrained Default Logic within KDL by giving a translation of Constrained Default Theories to K-default theories. A similar interpretation can be defined for Brewka's CDL.

In constrained default logic the language and the notion of default theory are the same as in Reiter's default logic. Let us recall the quasi-inductive definition of constrained extension in [Schaub92].

Definition ***(constrained extension)***. Let (D,W) be a default theory. Define E_0=W, C_0=W and, for all i>0,

E_{i+1}= Th(E_i) ∪ {γ | (α : β / γ)∈ D, α∈ E_i and C∪{β,γ} is consistent}

C_{i+1}= Th(E_i) ∪ {β ∧ γ | (α : β / γ)∈ D, α∈ E_i and C∪{β,γ} is consistent}.

(E,C) is a *constrained extension* of (D,W) iff (E,C) = ($\cup_{i=0,\omega}$ E_i , $\cup_{i=0,\omega}$ C_i).

In the definition above Th(E_i) denotes the deductive closure of E_i in classical logic and also the consistency of C∪{β,γ} is checked in classical logic. Notice that, to enforce commitment to assumptions, the justifications and consequents of applied defaults are recorded in the context C of the extension. Consider again Example 1.

Example 1 (contd.):

D={:B/C , :¬ B/ D}, W={}.

(D,W) has two constrained extensions (Th({C}), Th({C,B})) and (Th({D}), Th({D, ¬B})). On the contrary, we have seen in Section 2 that (D,W) has a single Reiter extension in which C and D are both true.

In [Schaub91b] a notion of lemma default rule is introduced, by means of which cumulativity in constrained default logic is preserved. Moreover, constrained default logic is semi-monotonic and existence of extensions is guaranteed.

We will now give an interpretation of constrained default logic in KDL by defining a mapping from default theories to K-default theory which preserves constrained extensions. As explained above, we will make use of modal operators to keep track of the context.

Definition ***(modal interpretation for constrained default logic).*** Let (D,W) be a default theory. An associated K-default theory (D_S,W_S) can be defined as follows:

W_S = W ∪ □W and

D_S= { A:◇(B∧C) /(C∧□ C∧□ B) | (A:B/C) ∈ D },

where, given a set of formulas Γ, □Γ= { □α | α ∈ Γ }.

Note that □ B occurs in the conclusion of the modal default rule and this allows to *commit* to the assumption B. This is in perfect agreement with the interpretation given in [Besnard&Schaub92] to the notion of commitment, i.e. that commitments correspond to formulas whose necessity holds.

It is possible to prove that there is a one to one correspondence between the constrained extensions of a default theory (D,W) and the KDL extensions of the associated K-default theory (D_S,W_S).

Theorem. Let (D,W) be a default theory and (D_S,W_S) its associated K-default theory. Then, (E,C) is a constrained extension of (D,W) iff there is a KDL extension F of (D_S,W_S) such that

$$E = \{ \alpha \mid \alpha \in L \text{ and } \alpha \in F \} \text{ and } C = \{ \alpha \mid \alpha \in L \text{ and } \Box \alpha \in F \}.$$

The proof of this theorem can be done by making use of the quasi-inductive definitions of constrained extension and of KDL extension. Remember that L is the non modal part of the language L^*, hence E contains all the non-modal formulas in F and C contains all the non-modal formulas α such that $\Box \alpha$ is in F.

We have seen that the notion of KDL extension is quite similar to that of Reiter's extension and it does not involve recording justifications in a set of constraints, as in constrained default logic. To obtain the equivalence result above in the modal translation of the default theory (D,W) an appropriate use of modal operators in default rules has been required to allow justifications and consequents to be recorded: the modal formulas in the KDL extension play the role of the constraint C in the constrained extension.

Notice that if a default A:B/C is in D, then D_S contains a corresponding default whose justification $\Diamond(B \wedge C)$ explicitly contains C. This is due to the fact that in constrained default logic default rules are implicitly regarded as seminormal (i.e. of the form $A:B \wedge C/C$) and this must be made explicit in the modal translation.

A similar modal interpretation within KDL can be given to CDL, the cumulative variant of DL proposed by Brewka [Brewka91]. In fact, as already mentioned above, there are precise results of equivalence between constrained default logic and CDL [Schaub92]. Since CDL allows *assertions* (i.e. formulas with a support) in the language, in order to give a modal interpretation to default theories in CDL, it suffices to find a suitable modal translation for assertions. The modal translation of default rules in CDL, instead, is the same as the one given above for constrained default logic. See [Giordano92] for details.

4. Mapping Lukaszewicz' default logic to KDL

The alternative formalization of default logic proposed in [Lucaszewicz88] is motivated by the need of having a system in which existence of extensions and *semimonotonicity* (i.e. monotonicity with respect to default rules) are guaranteed. This variant employs the following modified criterion of default applicability [Lucaszewicz88]: "If the prerequisite of a default is believed (its justification is consistent with what is believed), and adding its consequent to the set of beliefs neither leads to inconsistency nor contradicts the justification of this or any other already applied default, then the consequent of the default is to be believed".

Lucaszewicz defines the notion of m-extension (modified extension) of a default theory essentially as a pair (E,J), in which E is concerned with beliefs derivable from

the theory, while J is used to keep track of justifications supporting those beliefs (the approach is similar to the one followed by Schaub). m-extensions can be defined in the following way (see [Lucaszewicz88]).

Definition ***(m-extension)***. Let (D,W) be a default theory. Define $E_0=W$, $J_0=\emptyset$ and, for all i>0,

$E_{i+1}= Th(E_i) \cup \{\gamma \mid (\alpha : \beta / \gamma) \in D,\ \alpha \in E_i$ and, for each $\eta \in J \cup \{\beta\}$ $E \cup \{\eta,\gamma\}$ is consistent$\}$

$J_{i+1}= J_i \cup \{\beta \mid (\alpha : \beta / \gamma) \in D,\ \alpha \in E_i$ and, for each $\eta \in J \cup \{\beta\}$, $E \cup \{\eta,\gamma\}$ is consistent$\}$.

(E,J) is a *m-extension* of (D,W) iff $(E,J) = (\cup_{i=0,\omega} E_i , \cup_{i=0,\omega} J_i)$.

Notice that in m-extensions J is not deductively closed, differently from the context C in constrained extensions. Moreover, in J only justifications of applied defaults are recorded and not consequents, and justifications of applied defaults are not required to be consistent altogether; they only must be individually consistent with each consequent of applied defaults. Since consistency between justifications is not checked, the default theory of Example 1, (D,W) with $D=\{:B/C\ ,\ :\neg B/ D\ \}$ and $W=\{\}$, has a single default extension $(Th(\{C,D\}), \{B,\neg B\})$, which corresponds to the single Reiter extension. To see the difference between Lukaszewicz' variant and default logic, consider the following enlarged default theory.

Example 2.

$D=\{:B/C\ ,\ :\neg B/ D\ ,\ :\neg D \wedge \neg C / E\},\ W=\{\}$.

(D,W) has two m-extensions $(Th(\{C,D\}), \{B,\neg B\})$, and $(Th(\{E\}), \{\neg D \wedge \neg C\})$. However, (D,W) has a single Reiter extension $Th(\{C,D\})$. Notice that this theory has three constrained extensions: $(Th(\{C\}), Th(\{C,B\}))$, $(Th(\{D\}), Th(\{D, \neg B\}))$ and $(Th(\{E\}), Th(\{E,\neg D,\neg C\}))$.

We will now define an interpretation of Lukaszewicz' default logic in KDL.

Definition ***(modal interpretation for Lukaszewicz' default logic)***. Let (D,W) be a default theory. An associated K-default theory (D_L,W_L) can be defined as follows:

$W_L = W \cup \Box W$ and

$D_L= \{\ A:\Diamond B \wedge \Box C / (C \wedge \Box\ C \wedge \Diamond\ B)\ \mid\ (A:B/C) \in D\ \}$.

As a difference with previous mappings, in this case $\Diamond B$ is put in the consequent of default rules instead of $\Box B$. This is because in this case there is no commitment to assumptions. Hence, defaults with inconsistent justifications like $\Diamond B$ and $\Diamond \neg B$ can be applied together. Notice also that $\Box C$ in the justification of default rules in D_L is needed to guarantee that C is consistent with all justifications. Consider again Example 1. Its interpretation in KDL is the following

$D_L=\{:\Diamond B \wedge \Box C / (C \wedge \Box\ C \wedge \Diamond\ B)\ ,\ :\Diamond \neg B \wedge \Box D / (D \wedge \Box\ D \wedge \Diamond \neg B)\ \}$

$W_L=\{\}$.

(D_L,W_L) has a single KDL extension

$$E=Th_K(\{C, \Box C, \Diamond B, D, \Box D, \Diamond \neg B\}),$$

corresponding to the single m-extension $(Th(\{C,D\}), \{B,\neg B\})$. Notice that E contains both $\Diamond B$ and $\Diamond \neg B$, and it is K-consistent.

Also for Lukaszewicz' variant it is possible to prove that there is a one to one correspondence between the m-extensions of a default theory (D,W) and the KDL extensions of the associated K-default theory (D_L,W_L).

Theorem. Let (D,W) be a default theory and (D_L,W_L) its associated K-default theory. Then, (E,J) is a constrained extension of (D,W) iff there is a KDL extension F of (D_L,W_L) such that

$E = \{ \alpha \mid \alpha \in L \text{ and } \alpha \in F \}$ and

$J = \{ \alpha \mid \alpha \in L, \Diamond \alpha \in F \text{ and } \Diamond \alpha \text{ occurs in the justification of some } d \in GD(F)\}$,

where $GD(F)= \{ (A:B/C) \in D_L \mid A \in F \text{ and } \neg B \notin F \}$ is the set of generating defaults of F in (D_L,W_L).

Hence, J contains all α such that $\Diamond \alpha$ is in F and it occurs in the justification of a generating default *d* of F.

As regards Reiter's default logic, we have mentioned in section 2 that there is a straightforward mapping of default theories to KDL which preserves Reiter's extensions. It is the identity mapping, which maps a theory (D,W) to a theory $(D_R,W_R)=(D,W)$. Note, however, that if we modify the modal interpretation given above for Lukaszewicz' default logic, by cancelling $\Box C$ from the justification of default rules in D_L, we get the following alternative ***modal interpretation for Reiter's DL***:

$$W_R = W \cup \Box W \quad \text{and}$$

$$D_R= \{ A:\Diamond B /(C \wedge \Box C \wedge \Diamond B) \mid (A:B/C) \in D \}.$$

As for Lukaszewicz' variant, also in this case there is no commitment to assumptions; hence $\Diamond B$, and not $\Box B$, is put in the consequent of default rules. Differently from it, however, $\Box C$ is not included in the justification. In fact in Reiter's default logic inconsistent justifications are allowed and the consistency of default consequent is not required in order to apply a default.

Also for this second modal interpretation of Reiter's default theory it is possible to prove that there is a one to one correspondence between the Reiter's extensions of a default theory (D,W) and the KDL extensions of the associated K-default theory (D_R,W_R).

5. Commitment to assumptions

The modal interpretations proposed above for the different variants of default logics mainly differ as regards the interpretation of default rules. Let us summarize the

different translations. A default rule d= A:B/C can be translated into a KDL default rule in the following ways:

$d_S=d_B$= A:◇(B∧C) /(C∧□ C∧□ B) Constrained default logic and CDL,

d_L= A:◇B∧□C /(C∧□ C∧◇ B) Lukaszewicz' variant,

d_R= A:◇B/(C∧□ C∧◇ B) Reiter's default logic.

We have already mentioned that, on a syntactic ground, the presence of □B in the consequence of the translated default is what distinguish the logics that commit to assumptions (i.e. constrained default logic and Brewka's CDL) form those that do not. Moreover, unlike DL, constrained default logic and CDL not only commit to assumptions, but also regard default rules as seminormal, since the consequence C of *d* also occurs in the justification of d_S and d_B.

What kind of variant is obtained by committing to assumptions without regarding defaults as seminormal? Consider the following modal interpretation for the default *d*:

d_{CA}= A:◇B /(C∧□ C∧□ B).

This default rule allows commitment to the assumption B, since □ B is introduced in the consequent, while the consequent C is not checked for consistency.

In [Giordano&Martelli92] a cumulative variant of default logic, called CA-default logic (for *commitment to assumptions* default logic), has been proposed which has precisely this modal interpretations for default rules. This variant has been defined in the style of Brewka's CDL and, like CDL, it requires joint consistency of justifications. Unlike them, however, it does not consider default rules as being seminormal. Hence, it happens to be closer to Reiter's default logic than other cumulative variants. In particular, it does not guarantee existence of extensions and it is not semimonotonic. As such it allows priorities between defaults to be expressed, contrary to Brewka's CDL (see [Brewka91] section 3).

6. Conclusions

In this paper a uniform interpretation of different variants of default logics has been given by mapping them into a modal default logic KDL. KDL has the same definition as Reiter's default logic, but it is based on an underlying modal logic, K, instead of classical logic.

As mentioned in the introduction, a uniform semantical framework has been introduced by Besnard and Schaub [Besnard&Schaub92] for the variants of default logics considered in the previous sections. This semantics is defined in the same style as Etherington's default logic semantics [Etherington87] and as the semantics for CDL in [Schaub91a], but it makes use of Kripke structures. It must be noted that, for the different default logics, there is a precise correspondence between their Kripke semantics, as defined by Besnard and Schaub, and their translation within K-default logic given in the previous sections.

Acknowledgements

I want to thank Alberto Martelli for his helpful suggestions. This paper has been partially supported by CEC, in the context of the Basic Research Action, Medlar II.

References

[Besnard&Schaub92] P.Besnard and T. Schaub, Possible Worlds Semantics for Default Logic, in *Proc. Canadian Conference on AI*, 1992.

[Bowen79] K. Bowen, *Model Theory for Modal Logics*, Synthese Library, Reidel, Dordrecht, 1979.

[Brewka91] G. Brewka, Cumulative Default Logic: in defense of nonmonotonic inference rules, *Artificial Intelligence* 50: 183-205, 1991.

[Delgrande&Jackson91] J. Delgrande and W. Jackson, Default Logic Revised. In J.Allen, R. Fikes and E. Sandewall, eds., *Proc. KR'91*, pp. 118-127, Morgan Kaufmann, 1991.

[Etherington87] D. Etherington, A Semantics for Default Logic, in *Proc. Int. Joint Conf. on Artificial Intelligence*, pp.495-498, 1987.

[Hughes&Cresswell68] G.E. Hughes and M.J. Cresswell, *An Introduction to Modal Logic*, Methuen, London, 1968.

[Lucaszewicz88] W. Lucaszewicz, Considerations on Default Logic - an Alternative Approach, *Computational Intelligence*, 4:1-6, 1988.

[Makinson89] D. Makinson, General Theory of Cumulative Inference, in M. Reinfrank, eds., *Proc. Int. Workshop on Non-Monotonic Reasoning*, vol. 346 Lecture Notes in Artificial Intelligence, pp.1-18, Springer Verlag, 1989.

[Giordano92] L. Giordano, Defining Variants of Default Logic: a Modal Approach, Technical Report, Università di Torino, 1992.

[Giordano&Martelli92] L. Giordano and A. Martelli, On Cumulative Default Logics, *submitted.*

[Poole89] D. Poole, What the Lottery Paradox Tells us about Default Reasoning, in R. Brachman, H. Levesque, and R. Reiter, eds., *Proc. KR'89*, pp. 333-340, Morgan Kaufmann, 1989.

[Reiter80] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, 13:81-132, 1980.

[Schaub91a] T. Schaub, Assertional Default Theories: A Semantical View. In J.Allen, R. Fikes and E. Sandewall, eds., *Proc. KR'91*, pp. 496-506, 1991.

[Schaub91b] T. Schaub, On Commitment and cumulativity in Default Logics, in R. Kruse, ed., *Proc. European Conference on Symbolic and Quantitative Approaches to Uncertainty*, pp.304-309, Springer, 1991.

[Schaub92] T. Schaub, On Constrained Default Theories, in *Proc. 10th European Conference on Artificial Intelligence*, pp.304-308, Vienna, August, 1992.

An Admissible Heuristic Search Algorithm

Li-Yen Shue
Reza Zamani

Dept of Business Systems Uni of Wollongong
Australia

Abstract. This paper introduces an admissible heuristic search algorithm - Search and Learning Algorithm (SLA*). SLA* is developed from the work presented by Korf in the Learning-Real-Time-Algorithm (LRTA*). We retain the major elements of Korf's work in LRTA*, and improve its performance by incorporating a review component to fully reflect the effect the learning of new heuristic from front states has upon the previous states. The combined strategy of search, learning, and review has enabled this algorithm to accumulate knowledge continuously through guided expansion, and to identify better search directions in any stage of nodes expansion. With the assumption of non-overestimating initial estimates for all nodes to the goal, this algorithm is able to find an optimal solution in a single problem solving trial with good efficiency. We provide a proof for the optimality of the solution.

1 Introduction

Among the optimal heuristic search algorithms for graph problems, the most well known ones are A*[1] and IDA*[2], Iterative-Deepening-A*. A* is a best-first search algorithm, where the heuristic function of a node, f(n), is the sum of the actual cost in reaching that node from the root state, g(n), and the estimated cost of reaching the goal state from that node, h(n). As soon as a node is selected for expansion, this algorithm adds all its succeeding nodes to the selection list. In any stage of state selection, all nodes in the list have to be considered. The node with the minimum heuristic estimate is selected for expansion. One of the immediate drawbacks of this algorithm is the exponential growth of memory space requirement. The IDA* was designed with the intension to reduce the space complexity of the A*. Starting with the estimated initial threshold for the root state, this algorithm tries to find the next threshold by performing a series of depth-first searches. The minimum estimated value, f(n)=g(n)+h(n), of an iteration that exceeds the current threshold becomes the new threshold for the next iteration. With the assumption of non-overestimating initial estimates and positive edge costs between nodes, this algorithm will have its threshold increased in each

iteration, and reach the optimal solution in the end. The nature of this algorithm in focusing on finding the next threshold level, with no need to remember those nodes to be visited next time, does lead to the reduction of space complexity from exponential to linear. However, the repetitive search for the next threshold from the root node will lead to the same drawback as the A* algorithm in requiring exponential time to run in practice.

Another optimal heuristic search algorithm is the LRTA*[3], Learning Real Time Algorithm. This algorithm differs from the previous two methods in that this algorithm adapts a limited search horizon before making a decision move and the heuristic estimate to the goal of the visited node may be improved as search continues. The search horizon consists of all neighbouring nodes of a front node. The justification in improving the heuristic estimate of a node is based on the fact that a node's heuristic estimate to the goal must be at least as large as the minimum of its neighbours'. With the assumption of non-overestimating initial heuristic estimates and positive edge costs between nodes, the repetitive applications of the problem solver will lead to the optimal solution as the effect of the edge costs finally prevail. This algorithm presents the obvious advantages in both space complexity and time complexity over the previous two algorithms, although there is no guarantee of optimal solution in any single solution trial, nor is there any indication of how many solving trials are needed to reached an optimal solution..

In this paper, we introduce an admissible heuristic search algorithm - Search and Learning Algorithm (SLA*). This algorithm utilises heuristic as a search vehicle as others and learns from the comparison of heuristic estimates of a node's neighbours as the LRTA*. With the introduction of a review component and the application of the combined strategy of search, learning, and review, this algorithm is able to search for a new front state, and review the validity of its previous states and their respective heuristic estimates if a learning has occurred. As a result, this algorithm is able to maintain its states and their heuristic values up-to-date to account for full effect of heuristic learning during the search process, and find an optimal solution in a single problem solving trial.

2 Search and Learning Algorithm

SLA* algorithm works with the usual assumption that initial heuristic estimate of every state to the goal is a lower bound on its actual value. At a front state, to search for the next state for expansion, it compares the heuristic values of its neighbouring states and pick the one with the minimum value, as does the LRTA*. The same rationale used in LRTA* is also adapted in determining if a front state's heuristic estimate can be improved and by how much. In case, the indication is that an improvement can be made, the review component will be invoked immediately after the adjustment is made. The front state indicator of the algorithm will then point to its immediate previous state as the new front state, and a fresh evaluation is carried out to see if the newly revised heuristic value of its

neighbour can help improve its own heuristic estimate to the goal and by how much. The same process will continue to review the previously selected states one by one in the reverse order, and stop at the first state whose heuristic estimate remains unchanged after the re-evaluation. Then, from this front state, the search part of the algorithm resumes. As a result, every time when a backtracking occurs, states with their heuristic values modified will be detached from the original path, and their new values will be used for the re-examination of their previous states. When the search part finally resumes, the algorithm may or may not choose the same states again, because some states' heuristic has been altered, and depending on the heuristic estimates of other states, the algorithm may choose to explore new search direction.

With k(x,y) representing the positive edge cost from state x to a neighbouring state y, this algorithm can be implemented in the following details :

step 0 : Apply a heuristic function to generate non-overestimating initial heuristic estimate h(x) for every state x to the goal state, and continue.

step 1 : Put the root state on the backtrack list called OPEN, and continue.

step 2 : Call top-most state on the OPEN list x. If x is the goal state, stop; otherwise continue.

step 3 : If x is a dead-end state, replace its h(x) with a very large value , remove x from OPEN list, and go back to step 2; otherwise continue.

step 4 : Evaluate k(x,y)+h(y) for all neighbouring state y of x, and find the state with the minimum value; break ties randomly. Call this state x', and continue.

Step 5 : If h(x) >= k(x.x')+h(x'), then add x' to the OPEN list as the top-most state and go back to step 2; otherwise continue.

step 6 : Replace h(x) with k(x,x')+h(x'), and continue.

step 7 : If x is not the root state, remove x from OPEN list, and continue; otherwise continue.

step 8 : go to step 2.

Step 3 of the algorithm was designed to take care of problems with dead-end states, where the goal is not yet found and no further expansion is possible. By assigning a large value to the heuristic estimate of a dead-end state will ensure no future visit to the same state.

To demonstrate the operation of this algorithm, we use the graph in Figure 1. In the graph, a state is represented by an alphbet, and the initial heuristic estimate to the goal from a state is given by the corresponding number; the first number in case of states with multiple numbers. The multiple numbers of a state represent the updated estimates through the repeated review and updating operation. To simplify the calculation, we assume the edge cost to be 1 for all states. Assuming the current front state is state c, the detailed partial operation is given below:

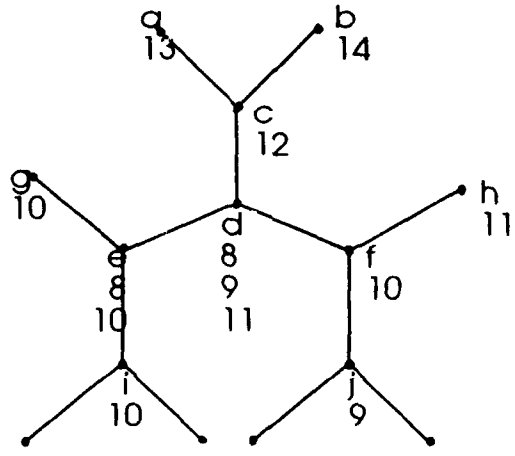


Fig. 1 Graph with initial & updated heuristic estimates returned by SLA*

At c, where h(c) = 12, min{[h(a)+1], [h(b)+1], [h(d)+1]} = {14, 15, 9} = 9. Since 9 < h(c), state d is selected as the new front state.

At d, where h(d) = 8, min{[h(c)+1], [h(e)+1], [h(f)+1]} = {13, 9, 11} = 9. Since 9 > h(c), h(c) is updated to 9, and state c becomes the new front state.

At c, where h(c) = 12, min{[h(a)+1], [h(b)+1], [h(d)+1]} = {14, 15, 10} = 10. Since 10 < h(c), state d is selected as the new front state.

At d, where h(d) = 9, min{[h(c)+1], [h(e)+1], [h(f)+1]} = {13, 9, 11} = 9. Since 9 = h(c), state e is selected as the new front state.

At e, where h(e) = 8, min{[h(d)+1], [h(g)+1], [h(i)+1]} = {10, 11, 11} = 10. Since 10 > h(e), h(e) is updated to 10, and state d becomes the new front state.

At d, where h(d) = 9, min{[h(c)+1], [h(e)+1], [h(f)+1]} = {13, 11, 11} = 11. Since 11 > h(d), h(d) is updated to 11, and state c becomes the new front state.

At this stage, the heuristic estimate of state d has been updated twice and once for state e, they appear to be more reasonable than before. As a result, there is no evidence to support any further improvement among states a, b, c, d, e, and f, any

further improvements have to be triggered by updating of other states. Should LRTA* be applied to the same problem, its first solving trial would have impoved state d to 9, and moved to state c to improve its value to 10, and then continue with states after that. It would need at least another solving trial from the root state to improve state d to 11.

3 Theorem and Proof

Theorem. For a finite problem space with positive edge cost and non-overestimating initial heuristic values, in which a goal state is reachable from the root state, the application of SLA* will find an minimum path.

Proof. Let P = X(1), X(2), ... X(n) denote the root-to-goal path returned by SLA*. Let P' = Y(1), Y(2), ... Y(t) be any other root-to-goal path. Let X(1) = Y(1) be the root state of a problem.

Let Y(m) be the first state of P' which is not on P. Thus both X(m) and Y(m) are neighbours of their previous common state X(m-1).

As indicated by step 5 and 6 of the algorithm, the following relation is always true for a state r under SLA*.

$$H\{X(r)\} >= K\{X(r),X(r+1)\} + H\{X(r+1)\} \quad (1)$$

This equation can be rearranged as

$$H\{X(r)\} - H\{X(r+1)\} >= K\{X(r),X(r+1)\} \quad (2)$$

By expanding and summing both sides of equation (2) over the state space of path P from m to the goal state, the following relation is obtained.

$$H\{X(m)\} - H\{X(n)\} >= K\{X(m),X(m+1)\} + K\{X(m+1),X(m+2)\} + \ldots + K\{X(n-1),X(n)\} \quad (3)$$

With the estimation from the goal state to itself being 0, $H\{X(n)\} = 0$, equation (3) is simplified to

$$H\{X(m)\} >= K\{X(m),X(m+1)\} + K\{X(m+1),X(m+2)\} + \ldots + K\{X(n-1),X(n)\} \quad (4)$$

The fact that SLA*, at state X(m-1), has preferred X(m) to Y(m), as indicated by step 4, has also led to the following relation.

$$K\{X(m-1),Y(m)\} + H\{Y(m)\} >= K\{X(m-1),X(m)\} + H\{X(m)\} \quad (5)$$

By substituting H{X(m)} of relation (5) with the right hand side of relation (4), relation (5) can be expressed as following:

$$K\{X(m-1),Y(m)\} + H\{Y(m)\} >= K\{X(m-1),X(m)\} + K\{X(m),X(m+1)\} + K\{X(m+1),X(m+2)\} + \ldots + K\{X(n-1),X(n)\} \quad (6)$$

It is obvious from this relation that by deviating from P at any state X(m-1), the estimate to the goal of the remaining path will always be greater than or equal to the true value of the corresponding path of P. With the non-overestimating nature of updating a state's heuristic in the algorithm, the true value of the former might even be much greater than the later. Thus, the application of SLA* will find an optimal solution.

4 Efficiency of SLA*

The fact that this algorithm needs to store the heuristic estimate for every state and remember the states on the path at any time has led to the upper bound of space requirement as n plus the number of states on the path, where n is the total number of states of a problem. The number of states on the list is usually only a small portion of n. In practice, however, the actual memory requirement could be lower, because usually there exists a function which computes the original heuristic estimates, and it is only necessary to store in memory those values which differ from those computed.

To derive time efficiency for the SLA*, we assume that all numbers used in solving a problem are positive integers. With this assumption, the worst indictive case for this algorithm is n*s, where s is the returned actual cost from the root state to the goal state. This worst case may happen when the initial heuristic estimates to the goal for all states are zero, no information for the algorithm to learn initially, and all edge costs from one state to its neighbours are assumed to be one. The later assumption will lead to only one unit improvement in each updating process, and the n*s figure is the worst case when every state has to be visited s times. In reality, the actual worst case will only be a portion of this figure n*s, because all states except the root state will have a smaller actual cost to the goal state than s, they do not need to be visited as many times. For most average problems, where initial information for heuristic estimation is obtainable, and edge costs may vary from one state to another, the number of state visits required before an optimal path is found could be greatly reduced.

We have experimented both LRTA* and SLA* algorithms with square grid problems, where a state is represented by a cell, and a path from the root to the goal is represented by the connection of chosen cells. The borders between cells serve to represent edge costs. To simulate problems where some states are dead-ends, the corresponding borders can be setup as barriers to bar from crossing. With 5 different square sizes 10, 15, 20, 25, and 30, and 4 percentages of randomly

assigned barriers 15%, 25%, 35%, and 45%, a total of 20 problems were tested. The SLA* found all optimal solutions in a single problem solving trial, while LRTA* required more than one solving trial and as a result more state visits as expected. The ratios of the number of state visits to find the optimal solutions between the two algorithms range from 3.7 to 28.7. The trend is clear that the larger the size of a problem and the barrier percentage the larger this ratio becomes.

5. Conclusion

The proposed SLA* algorithm improves greatly the efficiency of the original LRTA* by incorporating a review component, which is to reflect fully the impact the heuristic estimate modification of a front state has upon the previously selected states and their heuristic values. The search is guided by the continuous accumulation of changes of heuristic values of visited states and their effects. Along the search process, this algorithm keeps tracking the "best" states at any moment. Due to the continuous effect of heuristic updating and the associated backtracking, the "best" states may be changing from time to time. Hence, a state on the final path may need to be re-visited as many times as necessary to gradually adjust its heuristic value to its actual one. In addition, depending on the initial heuristic estimates, states not on the final path may or may not be visited. Even if they are visited, they may not be visited to the entent of revealing their actual values, this represents a great advantage of SLA* over LRTA*.

In comparison, the fundamental difference between SLA* and LRTA* lies in the ability of SLA* to review its selected states and their heuristic values and make appropriate adjustments whenever a learning is encountered in changing the heuristic estimate of a front state. LRTA* does make adjustments to front states, however, it has no capability to review the appropriteness of its previous states. In effect, LRTA* uses repetitive problem solving trials to carry out the essence of the review work, which is certainly not a very efficient way. Another difference is the fact that SLA* is applicable to problems where there exists at least one path from the root state to the goal state, it is not necessary to require every state to have a path to the goal as is required by the LRTA*.

References

1. P.E. Hart N.J.Nilsson and B. Raphael "A formal basis for the heuristic determination of minimum cost paths", IEEE Trans. Syst. Sci. Cybern. 4 pp 100-107, 1968.
2. R.E. Korf, "Depth First Iterative Deepening: An Optimum Admissible Tree Search", Journal of Artificial Intelligence, 27, pp. 97-100, 1985.
3. R.E. Korf, "Real Time Heuristic Search", Journal Of Artificial Intelligence" Vol 42, No 2-3, March, pp. 189-211, 1990.

Building an Expert System Language Interpreter with the Rule Network Technique*

Shie-Jue Lee and Chih-Hung Wu

Department of Electrical Engineering, National Sun Yat-Sen University
Kaohsiung, Taiwan 80424, e-mail: leesj,johnw@ee.nsysu.edu.tw

Abstract. Expert systems are increasingly prevailing in the fields of industry, business, and defense affairs. Expert system languages are key to the development, and play a decisive role on the quality, of expert systems. Three major components are required in an expert system language: knowledge representation, control, and developing tools. Current major expert system languages have their individual pros and cons, in terms of each component. We are developing a new expert system language which combines the advantages of these languages. Knowledge are expressed in the form of facts and rules which consist of predicates, without requiring rules to be Horn clauses. Facts may include variables and patterns are matched by unification. Control of execution is characterized by the recognize-act cycle of forward-chaining to reason about and answer user questions. Rules can be added and deleted dynamically. Friendly debugging facilities are provided. An interpreter, using the rule network technique, for the new language has been constructed, and test results show that the language is effective.

1 Introduction

Expert systems are increasingly prevailing in the fields of industry, business, and defense affairs. A good expert system language may facilitate and speed up the construction, and may improve the quality, of expert systems. Therefore, the importance of developing good expert system languages cannot be overemphasized.

Three major components are required in an expert system language: knowledge representation, control, and developing tools. Current major expert system languages ,such as LISP, Prolog, OPS5, and CLIPS, have their individual pros and cons, in terms of each component. We are developing a new expert system language and we try to combine the advantages of these languages mentioned above and avoid their disadvantages. Knowledge are expressed in the form of facts and rules which consist of predicates, without requiring rules to be Horn clauses. Facts may include variables and patterns are matched by unification. Control of execution is characterized by the recognize-act cycle of forward-chaining to reason about and answer user questions. Rules can be added and

* Supported by National Science Council under grant NSC 81-0408-E-110-02.

deleted dynamically. Various debugging commands are provided to help programmers to investigate the behavior of their programs throughout the development process.

An interpreter for the new expert system language has been constructed. It applies the rule network technique and was written in C-Prolog [5]. Some problems have been tested and the result shows that the new language is effective.

2 Features of the New Expert System Language

The new language has the following features:

1. Each rule is expressed in the form, $LHS \Longrightarrow RHS$ which is the same as in CLIPS[1].
2. A *LHS* or a *RHS* is a sequence of predicates, using commas or explicit *AND*s to concatenate them together. A predicate is a predicate symbol followed by a number of terms. Definitions for predicates and terms are the same as those in first-order logic [2]. For example, $father(john, mary)$ is a predicate indicating that John is Mary's father. Each predicate in a *LHS* is called a *pattern*, and each predicate in a *RHS* is called an *action*.
3. A fact is a predicate. Variables are allowed to appear in a fact.
4. A variable is a question mark (?) followed by a sequence of alphabetic or numeric characters in which the first character is alphabetic. A *constant* (individual constant, function constant, or predicate constant) is a sequence of alphabetic (including _) or numeric characters.
5. Execution is controlled by forward-chaining or backward-chaining.
6. It can provide answers to user queries: why and how. That is, it can explain why a conclusion is obtained and why a certain piece of information is needed in the process of inference.
7. Rules are allowed to be added or deleted dynamically through actions in the right hand side of a rule.
8. The implementation is based on the Rete algorithm with some modifications.
9. A rich set of debugging commands are provided for programmers to develop their programs. Almost all the debugging commands in CLIPS are offered.

Therefore, the following are legal rules for the language:

- $father(?x, ?y), ancestor(?y, ?z) \Longrightarrow ancestor(?x, ?z)$.
- $phase(choose_player), player_select(?x) \Longrightarrow retract(phase(choose_player)), retract(player_select(?x)), assert(phase(choose_player)), write(\text{"choose c or h."})$.
- $get(?LHS), get(?RHS) \Longrightarrow write(\text{"Having learned a newrule."}), assert(?LHS \Longrightarrow ?RHS)$.
- $learn_phase \Longrightarrow write(\text{"Type in a rule or a fact."}), read(?x), assert(?x)$.

Currently, our language interpreter provides forward-chaining only. The facilities for answering why and how are not implemented yet.

3 The Rule Network

In an expert system, matching each rule individually against all facts in each cycle would be clearly inefficient. Forgy [4] proposed the Rete algorithm to solve

this inefficiency problem, in which rules are compiled into the rule network which consists of the pattern network and the joint network. The algorithm was used in many expert system language interpreters, including OPS5 and CLIPS. In the pattern network, facts matched to patterns are stored in alpha memories. In the joint network, sequences of consistent facts matched to patterns are stored in beta memories. By maintaining the rule network, duplicate check for pattern matching and partial matches with old facts is avoided. The information contained in the rule network only needs to be recomputed when changes occur in the working memory. For example, if a set of patterns match two of three facts in one cycle, a check for pattern matching with these three facts won't be done again in the next cycle.

3.1 The Pattern Network

The process of determining which facts have matched which patterns is done in the pattern network. Tests for constant and relationship matches are done in pattern nodes, with one pattern node for each test. Each pattern is complied into a path made of a sequence of pattern nodes in the pattern network, with an alpha memory attached at the end of the path. Different patterns may share the same pattern nodes. A *pattern match* occurs when a fact has satisfied a single pattern in any rule without regard to variables in other patterns. That is to say, a pattern match corresponds to a path along which a fact has traveled successfully. When a fact travels successfully along a path, it is stored in the alpha memory attached to the path. For example, suppose we have a set of patterns $\{(car\ ford), (car\ ?x), (car\ \neg ford), (bike\ ?y)\}$ where $\neg$ is the negation symbol. Two pattern nodes N_1 and N_2 are created for the first pattern $(car\ ford)$, with N_1 including the following test:

the first field is equal to the constant car.

and N_2 including the following test:

the second field is equal to the constant ford.

Now we compile the pattern $(car\ ?x)$ and find that N_1 can be shared. The variable $?x$ matches anything in the second field of the income objects which pass through N_1 successfully. As the third pattern compiled, which shares N_1 with $(car\ ford)$, creates a pattern node N_3 which performs the following test:

the second field is not equal to the constant ford.

Finally, a pattern node N_4 is created for the pattern $(bike\ ?y)$ and contains the following test:

the first field is equal to the constant bike.

Suppose we have four facts in the working memory: $(car\ ford)$, $(car\ benz)$, $(car\ car)$, and $(bike\ ford)$, denoted by f1, f2, f3, and f4 respectively. Then the content of the alpha memory for each pattern is listed below.

- For pattern $(car\ ford)$: $\{f1\}$.
- For pattern $(car\ ?x)$: $\{f1, f2, f3\}$.
- For pattern $(car\ \neg ford)$: $\{f1, f2\}$.
- For pattern $(bike\ ?y)$: $\{f4\}$.

3.2 The Join Network

Each alpha memory node in the pattern network acts as one input to a node in the join network where comparison of variable bindings across patterns is performed to ensure that variables have consistent values. The nodes in the join network, referred as *join nodes*, have two inputs nodes: one from some alpha memory node and another from some join node. A join node contains matching tests for the content of a alpha memory and the set of partial matches that have matched previous patterns. A *partial match* for a rule is any set of facts which satisfy the rule's patterns beginning with the first pattern of the rule and up to the underlying pattern. The partial matches for a sequence of patterns are stored in the *beta memory* of a join node. The first join node performs tests for the first two patterns and the remaining join nodes test the partial matches in the beta memory of a previous join node and the content of the alpha memory of an additional pattern. Partial matches are stored in join nodes and passed to next join nodes for further partial matching. For example, suppose we have two rules containing the following LHSs:

LHS of Rule_1:	LHS of Rule_2:
pattern1: (?x allocation ?y)	pattern1: (?x allocation ?y)
pattern2: (car ?x)	pattern2: (car ?x)
pattern3: (bike ?x)	pattern3: (truck ?x)

Then the first join node of Rule_1 would perform a test like:

The value of the 1st field of the fact bound to the 1st pattern is equal to the value of the 2nd field of the fact bound to the 2nd pattern.

The second join node of Rule_1 would receive as input the set of partial matches from the first join node and the fact that matches the third pattern, and contains the test:

The value of the 2nd field of the fact bound to the 3rd pattern is equal to the value of the 2nd field of the fact bound to the 2nd pattern.

Assuming we have seven facts in working memory: (*car ford*), (*car benz*), (*car car*), (*benz benz*), (*bike benz*), (*benz allocation germany*), and (*ford allocation usa*), referred as f1, f2,...., f7 respectively. Then the beta memory of the first join node of Rule_1 contains the following set of partial matches:

{{(*ford allocation usa*),(*car ford*)},{(*benz allocation germany*),(*car benz*)}}

or

{{f7, f1}, {f6, f2}}.

And the beta memory of the second join node of Rule_1 contains the following set of partial matches:

{{(*benz allocation germany*),(*car benz*), (*bike benz*)}}

or

{{f6, f2, f5}}.

The resulting rule network is shown in Figure 1.

A rule satisfied by facts also has a partial match of all of the patterns of the rule. Such rules are called *activated rules*. All the activated rules are collected in a *conflict set*, to which the last join node of rules are directed. One of the rules

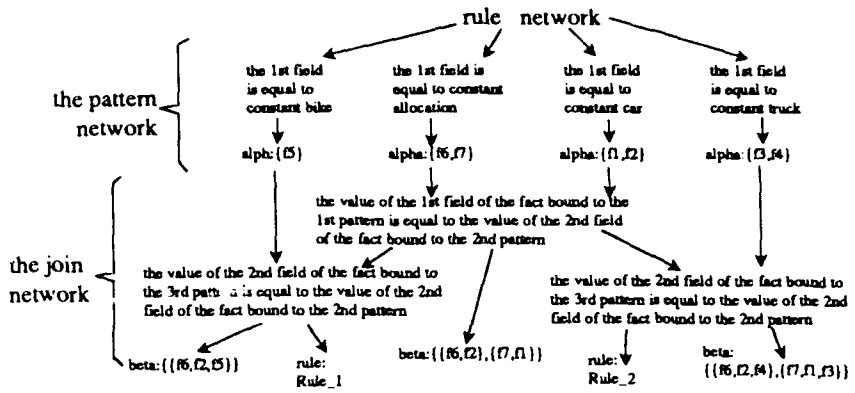


Fig. 1. The Rule Network of the rules Rule_1 and Rule_2

in the conflict set is selected according to *conflict resolution* policies and then is fired.

4 Algorithms

In this section, we describe some major algorithms used in the construction of the interpreter which executes expert system programs written in the new language. We use C-Prolog in the project. Unification for the matching between facts and patterns is done by Prolog itself. Besides unification, the interpreter has to take care of

1. Compiling rules into the rule network;
2. Performing the recognize-act cycle;
3. Interpreting facts to match patterns in the rule network;
4. Adding new facts/rules to the rule network or deleting existing facts/rules from the rule network dynamically;
5. Keeping the knowledge contained in the rule network consistent after addition/deletion of facts or rules;
6. Maintaining activated rules in the conflict set correctly;
7. Conflict resolution.

We describe how to add and delete rules dynamically, and conflict resolution in detail below. The other operations are simple and their descriptions are omitted here.

4.1 Addition of Rules

We use the command "insert" to perform the addition of rules and facts. Once the operator $\Longrightarrow$ is found in the argument of an insert command, we apply the Rete algorithm to compile the new rule into the rule network. If the new rule has structural similarity with existing rules, some of the nodes/paths/memories in the rule network can be shared by this new rule. A flag is set for facts rematching on those shared memory nodes. Moreover, new nodes and paths are created for new patterns and a flag is set for facts rematching on these newly created nodes and paths. The process of adding rules can be described as follows:

```
Algorithm: Addition of a Rule R
begin
      if R can share pattern nodes existing in the pattern network
      then use these pattern nodes instead of creating new ones;
      if some patterns P in R are existing in the pattern network
      then use the existing paths for P;
            use the existing alpha memory nodes A of these paths for P;
            if paths J of R are existing in the join network
            then use the paths J for R;
                  use the existing beta memory nodes of J for R;
            else creat new join paths J' in the join netwrok;
                 set a flag at A for facts rematching;
      else creat new pattern nodes/paths P' for R in the pattern network;
           creat new join nodes/paths for R in the join network;
           set a flag at P' for facts rematching;
      call the procedure "facts rematching" on J' and P';
end.
```

4.2 Facts Rematching

Facts rematching starts from alpha memory or beta memory nodes, whose flag is set, if the new rule shares memory nodes with some existing rules due to structural similarity. All of the facts/partial matches in a alpha/beta memory node M perform *tests in node* N, *which is a* successor node of M, with objects coming from another predecessor node. If partial matches occur in N, the tests in node P, which is a successor node of N, are performed on the partial matches and the objects coming from another predecessor node of P, and so forth. The algorithm is also called when new paths and their associated alpha memory nodes have been created. In this case, all of the facts in working memory are interpreted for pattern matching from the top node of the rule network along the new paths. Facts are stored in alpha/beta memory nodes if pattern matches/partial matches occur. Note that the new rules which have complete partial matches, during facts rematching, cannot be put into the conflict set. The procedure can be described as follows.

```
Algorithm: Facts Rematching
begin
      S ← {};
      while a flag is set at shared node N with memory node M
            S ← S∪{(N, M)};
      while a flag is set at newly created alpha memory nodes
            S ← S∪{(the top node of the rule network, the set of facts)};
      for all elements (Start, Bank) in S do
          for all elements E of Bank do
              for all successors S1 of Start do
                  if S1 is the last join node of some rule
                  then return;
                  if pattern matches occur at the successors S1 of Start
                  then perform pattern matching/partial matching, by E,
                                  on all successors of S1;
                  if partial matches occur on the successors S1 of Start
                  then perform partial matching, by E, on all successors of S1;
              end;
          end;
      end;
end.
```

4.3 Deletion of Rules

Rules can be deleted by their names dynamically. When a rule is retracted, the nodes/paths and alpha/beta memory nodes associated with this rule are removed from the rule network. Nodes in the rule network maintain two kinds of links, *one pointing to its successor nodes* and the other pointing to its predecessor node. Due to structural similarity, nodes in the rule network may be shared by lots of successor nodes. Therefore, they may have one or more successor links. Only one predecessor node exists in each node. When a rule R is deleted from the knowledge base, the activated version of R in the conflict set and the last join node L of R are deleted. Let J be a predecessor of L. If J has L as its only successor, then J is also deleted from the rule network. However, if J has two or more successor links, then only the successor link pointing to L is removed from J. The same process is performed on all of the predecessor nodes of J, and so forth. Here is the algorithm.

```
Algorithm: Deletion of a Rule R
begin
    S ← the last join node of R;
    F ← the successor node of S pointing to R;
    if activated versions Ra of R exist in the conflict set
    then delete all Ra's;
    while there is a node K in S do
        if the number of successors of K is equal to 1
        then remove K and its memory node;
            S ← S−{K} ∪ {x|x is a predecessor node of K};
            F ← K;
            return;
        else remove the link pointing to F;
            return;
    end;
end.
```

5 Conflict Resolution

Activated rules are collected in the conflict set and only one rule is selected and fired in each cycle. There are several policies that are considered when selecting one rule to be fired:

1. Rules with the largest number of condition patterns are fired first
2. Rules with the largest number of negated condition patterns are fired first
3. Rules with the most recently matched LHSs are fired first
4. New rules are fired first

The first policy has the highest priority, and the last policy has the lowest priority. If two or more rules are left after applying these policies, then one is selected arbitrarily for firing.

6 The Interpreter

An interpreter has been constructed to execute programs written in the new expert system language. It basically performs recognize-act cycles until no rules

can be fired, like OPS5 [3]. First of all, all rules of an input program are compiled into a rule network. Then facts in the working memory are interpreted from the root node of the pattern network and matched facts are stored in different alpha or beta memory nodes. Activated rules are pushed into a conflict set. Then the following operations are iterated until no rules can be fired.

1. Select one activated rule, according to the conflict resolution policies, from the conflict set and fire it.
2. Execute the actions specified in the right hand side of the fired rule.
3. Facts and rules are added to or deleted from the knowledge base and the rule network, as specified in the actions of the fired rule.
4. Due to the addition/deletion of facts/rules, the content of the memory nodes in the rule network have to be updated, new activated rules may be added to the the conflict set, and some rules in the conflict set may be deactivated.

The interpreter was written in C-Prolog and includes the following modules:

1. the rule network compiler.
2. fact interpreter.
3. conflict resolution.
4. action performing.

Figure 2 shows the block diagram of the interpreter.

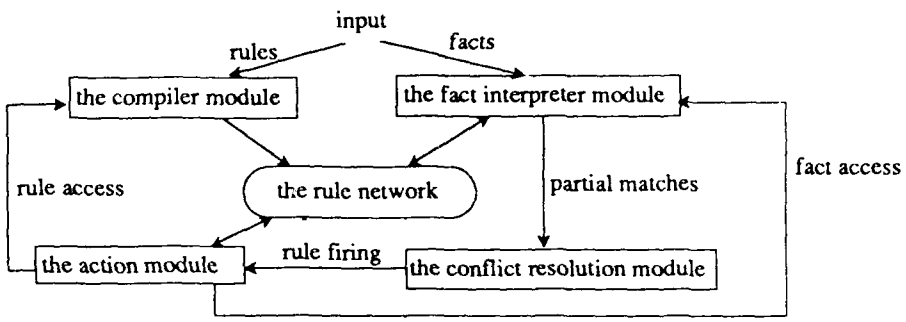


Fig. 2. The block diagram of the interpreter.

7 Test and Comparison

Two examples are given in this section. The first example tests that the interpreter works correctly. The second example shows the system's capability of changing rules dynamically and consistently.

7.1 Solving the Monkey-Banana Problem

The Monkey-Banana problem is described as follows: *A monkey is standing on a couch near the coordinate (5,7). There are bananas highly allocated near the coordinate (8,2). A light ladder is near the coordinate (2,2). The goal is : Let*

the monkey have the bananas. The problem can be translated to 8 facts and 19 rules. The rules are compiled into a rule network with 91 nodes. The solution of the problem consists of the following steps:

1. The monkey jumps off of the couch.
2. The monkey walks from [5,7] to [2,2].
3. The monkey grabs the ladder.
4. The monkey walks from [2,2] to [8,2].
5. The monkey drops the ladder.
6. The monkey climbs onto the ladder.
7. The monkey grabs the bananas.

We got the solution by our system and by CLIPS respectively on a DEC5000/125 workstation with 16MB memory.

Our system. Fifteen rules had been fired, i.e. 15 cycles had been executed, before the problem was solved. The total cpu time is 4.41667 seconds obtained by running C-Prolog (version 1.5) [5].

CLIPS. Twenty-three rules were fired, i.e. 23 cycles were executed. CLIPS applies only one resolution policy, requiring more cycles for this example. The total cpu time is 0.2 seconds. Note that CLIPS ran much faster than system, partly because the interpreter of CLIPS is written in C while our interpreter is written in C-Prolog which is much slower.

7.2 Dynamic Addition/Deletion of Rules

For the sake of comparison with CLIPS, we present this example in CLIPS format. Suppose we have a rule

(defrule rule001 (ford usa) (ford makes ?x) (?x good) $\Longrightarrow$
    (printout t "usa " ?x "are good." crlf) (assert (?x cheap)))

and facts: (ford usa) (ford makes cars) (ford makes truck) (cars good)

Suppose we want to add the following new rule after the first rule is fired:

(defrule rule002 (ford usa) (ford makes ?x) (?x good) (?x cheap) $\Longrightarrow$
    (printout t "usa " ?x "are good and cheap." crlf))

Let's see how CLIPS and our system work on this problem.

CLIPS. CLIPS does not allow us to do the insertion of the rule rule002 by putting it in the right hand side of the rule rule001. So we have to insert rule002 either manually or by loading a file containing rule002. After the rule rule002 is inserted, obviously there is a partial match from pattern 1 to pattern 3 for this rule since (cars cheap) has been added to the knowledge base after the rule rule001 was fired. However, this partial match is missing. CLIPS only maintains the partial match from pattern 1 to pattern 2 for the rule rule002. Therefore, rule002 is neither activated nor fired. Clearly, the rule network CLIPS maintains is not consistent.

Our system. Our system allows the problem to be described as follows:

rule(defrule rule001 (ford usa) (ford makes ?x) (?x good) $\Longrightarrow$
    (printout t "usa " ?x "are good." crlf) (getname ?name)
    (insert (defrule ?name (ford usa) (ford makes ?y) (?y good) (?y cheap)

```
    ⟹
    (printout t "usa " ?y " are good and cheap." crlf)))
  (insert (?x cheap))).
```

This rule is activated and fired. Assuming ?name is bound to rule002. Then the rule rule002 in the action part of rule001 is inserted and compiled into the rule network. After the compilation, the last action of rule001 is performed and (cars cheap) is inserted into the knowledge base and the rule network is updated. We can see that rule002 has the following 3 partial matches: {(ford usa), (ford makes cars)}, {(ford usa), (ford makes cars), (cars good)}, {(ford usa), (ford makes cars), (cars good), (cars cheap)}.

Since (cars cheap) is more recent than the rule rule002, rule002 is fired. Therefore, no inconsistency exists in our rule network and it seems that our result is more desirable.

8 Conclusion

We have proposed a scheme of a new expert system language. The language is rule-based. Patterns in the left hand side and the actions in the right hand side of a rule are predicates. Facts are predicates and allow the occurrence of variables. Patterns are matched by unification. Execution is controlled by forward-chaining. Rules are allowed to be added or deleted dynamically. A rich set of debugging commands are provided for programmers to develop their programs. Unification instead of one-way matching is used for pattern matching.

An interpreter, based on the rule network, was constructed to execute programs written in this new language. The interpreter was written in C-Prolog. Rules are compiled into a rule network for efficient pattern matching. Paths/nodes are created and joined into the rule network when new rules are added to the knowledge base, and are removed from the network when rules are deleted from the knowledge base. Facts rematching is performed along newly created paths in order to keep the knowledge contained in the rule network consistent. Added/Deleted facts may cause the content of alpha/beta memories to be updated. In each cycle, one rule is selected and fired, and its action part is performed.

References

1. Artificial Intelligence Section, Lyndon B. Johnson Space Center. *CLIPS User's Guide, Reference Manual, and Architecture Manual*, May 1989.
2. C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.
3. C. L. Forgy. *On the Efficient Implementation of Production Systems*. PhD thesis, Carnegie-Mellon University, 1979.
4. C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.
5. F. Pereira. *C-Prolog User's Manual*. SRI International, Menlo Park, California.

Input-Driven Control of Rule-Based Expert Systems

Gabriel Valiente Feruglio*

Universitat de les Illes Balears
Dept. de Ciències Matemàtiques i Informàtica
E-07071 Palma (Balears) Spain

Abstract. Most expert system control strategies focus on a narrow view of problem solving, with the resulting expert systems failing to accomplish a good interaction with the task environment during problem solving. Known as the *pregnant man* problem, such a deficient external behavior is typical of many expert systems that ask absurd questions of the user. Right interaction with the task environment has been already addressed as a verification problem. A new framework for realizing a good external behavior is presented in this paper, that consists of incorporating constraints on *external behavior in the control strategy* of the expert system. Called *input-driven control*, it cooperates with the standard control strategies of the Milord II expert system shell for realizing *both* a correct problem-solving behavior and an appropriate external behavior.

1 Introduction

Expert systems attempt to reproduce the intelligent problem-solving behavior exhibited by human experts in ill-defined problem domains. A sharp boundary can be drawn between two aspects of such an intelligent behavior, the one being solving problems right and the other dealing with attaining an appropriate interaction with the task environment during problem solving.

Current approaches for developing expert systems take a somewhat narrow notion of right solutions to problems. Consider for instance a diagnosis expert system. Given a set of symptoms, any hypothesis entailed by the knowledge base (enlarged with the symptoms) is taken as a right solution, no matter *how* the hypothesis has been proved to be a logical consequence of the (enlarged) knowledge base. The counterpart of such a notion in logic is that of *entailment*.

It may happen, and is most often the case, that these symptoms are not given beforehand. The expert system has to interact with the task environment in order to get the (external) data. Even when all the relevant symptoms are known beforehand, additional external data may be needed during the consultation process in order for the expert system to arrive at a diagnosis.

Right interaction with the task environment is so important from the standpoint of expert system usability and acceptability [2] [4] that failing to accomplish a good interaction with the task environment during problem solving has

* Partially supported by grant DGCIyT PB91-0334

been recognized as the *pregnant man* problem, typical of many expert systems that ask absurd questions of the user. Asking a question about whether the subject is a pregnant before asking a question about the sex of the subject is the classical example of bad interaction with the task environment, because the question about whether the subject is a pregnant makes no sense in the case of a male subject.

The previous discussion leads to a wider notion of right solutions to problems, namely those solutions that, besides being logical consequences of the knowledge base, have been obtained through a right interaction with the task environment. The counterpart of such a notion in logic is that of *proof system*.

The use of the weaker notion of solution is, in fact, rooted in a problem inherent to the self nature of the rule-based paradigm. Although the rule-based paradigm has been offered as a declarative programming tool, expert system developers have to *think procedurally* in order to know that the procedural reading of the knowledge base by the inference engine will consider premises in a rule in the, say, left-to-right order in which they have been written, and write therefore a rule like *if female and ... and pregnant and ... then ...* for getting the expert system ask a question about the sex of the subject before asking a question about whether the subject is a pregnant.

This issue becomes ever more complicated under more complex control strategies [8] [11], such as those involving the modular structure of the knowledge base, the premise ordering within each rule, the rule ordering within each module, rule certainties, rule specificities, etc. Expert system developers usually end up reasoning about the order of module calls, rule selection, premise selection, and the like. Right external behavior is only achieved, with very much effort, by trial and error: by experimentation with and modification of a prototype of the expert system until observing a right external behavior. The main problem with this approach is that it is not systematic. Even worse, it forces expert system developers to think procedurally, reasoning about the order of rule firing instead of concentrating themselves on achieving a good conceptualization of their problem domain.

Right interaction with the task environment has been already addressed as a verification problem [6] [5] [13] [14]. A new framework for realizing a good external behavior is presented in this paper, that consists of incorporating constraints on external behavior in the control strategy of the expert system.

The rest of the paper is organized as follows. The specification language being used, together with a computer-supported methodology aimed at acquiring correct and complete specifications of external behavior, is presented in section 2. The new control strategy, called input-driven control, is presented in section 3. Finally, some conclusions are drawn in section 4.

2 Problem solving and external behavior

The relationship between problem solving and external behavior can be best understood, by thinking of a consultation as a way from a state in which the

knowledge in the expert system is of a general nature (i.e. applicable to a wide class of problems) to a state in which the knowledge has been specialized to a particular problem —or to a more restricted class of problems— by the data provided by the task environment. This behavior being common to all expert system shells, it is most clear in the Milord II shell [10] [9], that uses such a specialization principle as the only inference rule.

Consider the following rules for pneumonia treatment, adapted from [9], where *H-Influenzae* and *Legionella-sp* are possible diagnoses and *Quinolones* and *Co-trimoxazole* are antibiotics.

($R0$) **if** H-Influenzae **then** Quinolones **is** possible
($R1$) **if** female **and** young **and** pregnant **and** Legionella-sp **then** Co-trimoxazole **is** slightly-possible
($R2$) **if** female **and** young **and** breast-feeding **and** Quinolones **is** possible **then** stop-breast-feeding **is** definite
($R3$) **if** breast-feeding **and** Co-trimoxazole **then** stop-breast-feeding **is** definite

Consider the case of a young female patient with a diagnosis of H-Influenzae. When the user is asked for the value of the diagnosis, suppose she answers *very-possible*. With this information from the task environment, the expert system specializes the knowledge base to the following.

($R1$) **if** female **and** young **and** pregnant **and** Legionella-sp **then** Co-trimoxazole **is** slightly-possible
($R2'$) **if** female **and** young **and** breast-feeding **then** stop-breast-feeding **is** definite
($R3$) **if** breast-feeding **and** Co-trimoxazole **then** stop-breast-feeding **is** definite

Next the user is asked for the sex of the patient, and answers *female*. The specialized knowledge base follows.

($R1'$) **if** young **and** pregnant **and** Legionella-sp **then** Co-trimoxazole **is** slightly-possible
($R2''$) **if** young **and** breast-feeding **then** stop-breast-feeding **is** definite
($R3$) **if** breast-feeding **and** Co-trimoxazole **then** stop-breast-feeding **is** definite

Finally the user is asked for the age of the patient, and answers *young*. The resulting knowledge base follows.

($R1''$) **if** pregnant **and** Legionella-sp **then** Co-trimoxazole **is** slightly-possible
($R2'''$) **if** breast-feeding **then** stop-breast-feeding **is** definite
($R3$) **if** breast-feeding **and** Co-trimoxazole **then** stop-breast-feeding **is** definite

This knowledge base represents a specialization of the more general ones for treatment of young female patients with a diagnosis of H-Influenzae.

Now two different readings of this consultation can be made, one centered on the hypotheses being pursued and the deductions being made, and the other centered on the interaction with the task environment —the questions that are asked of the user during problem solving and the order in which they are made. It is this second reading what is known as *external behavior*.

The ideal relationship between problem solving and external behavior is, of course, that of *cooperation*. This means that all questions asked of the user during the consultation are made only when they are relevant and necessary to the current problem-solving state, and that the answers given by the user determine the number and order of questions that follow. This is just the behavior that is usually thought of as *expert*.

Cooperation between problem solving and external behavior can be achieved by incorporating constraints on external behavior in the control strategy of the expert system. Specification concepts for external behavior are developed in the following.

Specification of external behavior

A general framework for specifying the intended external behavior of an expert system, consisting of both domain-dependent constraints on external behavior and domain-independent criteria for question selection, is proposed here. Domain-dependent constraints are divided into order relations over questions and question-relevance contraints.

Two different order relations over questions are defined: strong data dependency and weak data dependency [13] [14]. Note that the words question and external fact are often used here as synonymous. Strictly speaking, question refers to the act of asking the value of the attribute expressed by an external fact to the user.

A question q_j depends weakly on a question q_i, noted $q_i \prec q_j$, if q_j cannot be asked before q_i. A question q_j depends strongly on a question q_i, noted $q_i < q_j$, if $q_i \prec q_j$ and, furthermore, q_j cannot be asked unless q_i has been already asked. For instance, let *sex* be a question about the sex of the subject and let *pregnant* be a question about whether the subject is a pregnant. Then *sex* $<$ *pregnant* (and not just *sex* $\prec$ *pregnant*) because a question about whether the subject is a pregnant should by no means be asked unless the sex of the subject is known to be female (and this can only be known, for the sake of this example, by asking a question of the user about the sex of the subject).

Question-relevance constraints dig something deeper into question order by also considering answers (values) to questions. Similar constraints are present in some commercial expert system shells, such as the "presupposition" statement in Tecknowledge's M.1, although their evaluation may have undesirable side effects.

In the case of the Milord II shell [10] [9], question-relevance constraints are predicates of the form *q unless condition* where *condition* is a premise whose evaluation does not produce any side effect. That is, such conditions are just

tested against the current state of the execution, without originating module calls, rule firings, deductions, nor further questions asked of the user.

Returning to the previous example, a constraint like *pregnant unless sex = male* would be more appropriate than the strong data dependency *sex < pregnant*. (Discovering a data dependency in a problem domain is, however, a step in the direction of discovering a possible question-relevance constraint.)

It must be noted that the distinction made between question relevance and question order is, to a certain extent, relative, because at the very end question relevance and question order are two sides of the same coin. Pushing the notion of question relevance to the limit, an expert system that always asks the most relevant question will obviously ask all the relevant questions in the right order.

Domain-independent criteria for question selection are intended for covering those cases in which domain-dependent constraints on external behavior do not suffice for selecting *one* question to ask of the user next. These criteria serve therefore for further constraining the set of possible questions to be asked of the user. Typical domain-independent criteria include measures of cost, risk, etc. associated to each question (external fact) in the knowledge base, and, of course, the (lexicographical) order in which external facts have been declared in the knowledge base modules.

When confronted with the problem of defining constraints on question relevance and order for a relatively large expert system, no human expert would arrive at a complete specification without some systematic way of focusing attention on small sets of highly related questions. A methodology providing detailed step-by-step instructions for arriving at a complete (and correct) definition of the domain-dependent component of a specification of external behavior, has been developed that is based on the method of Herod and Bahill [6] [5] of question matrices, with some extensions for dealing with a large number of questions (coupling then with modular techniques underlying the base expert system language) and for defining a more comprehensive specification of external behavior. The methodology has been omitted here due to severe space limitations, and the reader is referred to [15] for a complete description.

The acquisition methodology is supported by a tool called QuestionnAIre, that has been implemented with a direct-manipulation interface for filling in the different question matrices. The tool can be used with knowledge bases written for the Milord II shell [10] [9] and is being applied to the development of ENS-AI [1], an expert system for assisting teachers to deal with pedagogical problems.

3 Input-driven control of rule-based expert systems

Control strategies are usually described in terms of the classical *recognize-act cycle* and the associated concept of conflict resolution [8] [11]. An alternative formulation, centered on question selection —and therefore more appropriate for studying external behavior aspects— replaces the recognize-act cycle by an equivalent ***specialize-ask cycle***. (The specialize-ask cycle does not preclude the use of conflict resolution criteria, however.)

The specialize-ask cycle underlies most partial evaluation architectures, such as that of the Milord II shell [10] [9]. Specialization is achieved by applying a specialization inference rule to the knowledge base that, given a proposition A with certainty value α and a rule of the form *if A and B then C* with certainty value ρ specializes it to a rule of the form *if B then C* with certainty value ρ', where ρ' is computed by *modus ponens* from α and ρ [9].

The *ask* step is performed following a *question selection* strategy. The standard question selection strategy in Milord II reproduces the conflict resolution criteria of the Milord shell [3] [7], namely rule certainty, rule specificity and (lexicographical) order of rules [12].

Question selection is called *heuristic* if it follows the specification of external behavior defined for the problem domain when deciding which question to ask next. Heuristic question selection has been implemented in Milord II as an input-driven control strategy that cooperates with the standard question selection strategy used in the Milord II shell.

Input-driven control requires a "constructive" interpretation of the specification of external behavior. This interpretation determines, given any intermediate problem-solving state and a set of potential questions to ask next, which question(s) satisfy the specification and can therefore be asked next. A question q can be asked next in any given problem-solving state if *all* the following conditions are satisfied:

- The question q has not been asked, i.e. it has not been assigned a value in the given problem-solving state;
- For all strong data dependencies of the form $p < q$, the question p has already been asked;
- For all weak data dependencies of the form $q \prec r$, the question r has not been asked yet; and
- For all question relevance constraints of the form q *unless condition*, *condition* is not satisfied by the current problem-solving state.

Notice that weak data dependencies of the form $p \prec q$ do not constrain the expert system asking question q next, but they would later constrain the expert system asking question p if question q is asked before.

The constructive interpretation of the specification of external behavior has been implemented as a cooperating component of the Milord II expert system shell [10] [9]. Cooperation is achieved as follows:

- KB specialization is performed as usual, by iterative application of the specialization inference rule until no further specialization is possible given the available external data.
- Question selection after the standard control strategy of Milord II, that ranks candidate rules according to certainties, specificities, and (lexicographical) order, prevails unless it violates the specification of external behavior. In such a case, heuristic question selection takes control for selecting the first question, according to the ranking just made by the standard control strategy, that does not violate the specification of external behavior.

- The selected question is asked of the user, and the specialize-act cycle continues until reaching a stop condition.

Notice that it may happen that no question can be selected in a given problem-solving state, that has not already been asked and does not violate the specification of external behavior. These situations are, in fact, nothing more than additional *stop conditions* of the expert system.

Cooperation between problem solving and external behavior, as presented in this section, corresponds then to a modified *ask* step in the specialize-ask cycle, by which only those questions satisfying *both* problem-solving requirements and external behavior requirements are asked of the user.

An example

The following module excerpt, taken from the Terap-IA expert system [9], has been developed using the conventional, trial-and-error approach for achieving an appropriate external behavior. It comprises metarules (i) to avoid asking a question about whether the patient is a pregnant or a breast-feeding to a man or to an elder woman, (ii) to avoid asking a question about whether the patient is a breast-feeding to a pregnant, and (iii) to avoid asking facts that depend on pregnant or breast-feeding if pregnant or breast-feeding do not hold.

```
Module anam =
Begin
 Import age, sex, pregnant, pregnant_term, pregnant_period, breast_feeding,
        neonate, premature
 Export ...
 Deductive Knowledge
  Dictionary:
   Predicates:
    age = ...
    sex = ...
    pregnant = ...
     relation: needs sex
     relation: needs age
    pregnant_term = ...
     relation: needs pregnant
    pregnant_period = ...
     relation: needs pregnant
    breast_feeding = ...
     relation: needs sex
     relation: needs age
     relation: needs pregnant
    neonate = ...
     relation: needs breast_feeding
    premature = ...
     relation: needs breast_feeding
    ...
```

```
   Rules:
    ...
 End deductive
 Control knowledge
  Deductive control
   M001 if T(pregnant,sure) then conclude T(not(breast_feeding),sure)
   M002 if is(=(sex,(male)),sure) then conclude T(not(pregnant),sure)
   M003 if is(=(sex,(male)),sure)
        then conclude T(not(breast_feeding),sure)
   M004 if is(=(sex,(female)),sure) and T(=(age,$x),sure) and lt($x,15)
        then conclude T(not(pregnant),sure)
   M005 if is(=(sex,(female)),sure) and T(=(age,$x),sure) and lt($x,15)
        then conclude T(not(breast_feeding),sure)
   M006 if is(=(sex,(female)),sure) and T(=(age,$x),sure) and gt($x,45)
        then conclude T(not(pregnant),sure)
   M007 if is(=(sex,(female)),sure) and T(=(age,$x),sure) and gt($x,45)
        then conclude T(not(breast_feeding),sure)
   M008 if T(not(pregnant),sure) then conclude T(not(pregnant_term),sure)
   M009 if T(not(pregnant),sure)
        then conclude is(=(pregnant_period,(none)),sure)
   M010 if T(not(breast_feeding),sure) then conclude T(not(premature),sure)
   M011 if T(not(breast_feeding),sure) then conclude T(not(neonate),sure)
  Evaluation type: lazy
 End control
End
```

The same module excerpt, rewritten to make use of heuristic question selection as implemented in the Milord II shell, is reproduced below.

```
Module anam =
Begin
 Import age, sex, pregnant, pregnant_term, pregnant_period, breast_feeding,
  neonate, premature
 Export ...
 Deductive Knowledge
  Dictionary:
   Predicates:
    age = ...
     relation: strong pregnant
     relation: strong breast_feeding
    sex = ...
     relation: strong pregnant
     relation: strong breast_feeding
    pregnant = ...
     relation: strong breast_feeding
     relation: strong pregnant_term
     relation: strong pregnant_period
    pregnant_term = ...
    pregnant_period = ...
    breast_feeding = ...
```

```
     relation: strong neonate
     relation: strong premature
    neonate = ...
    premature = ...
    ...
   Rules:
    ...
 End deductive
 Control knowledge
  Evaluation type: heuristic
 End control
End
```

Getting the right metarules for avoiding inadequate question sequences has proved to be no easy task. Leaving such methodological issues apart, the previous implementation scerpts illustrate the difference between the trial-and-error approach and the disciplined approach to external behavior presented in this paper. The improvement in declarativeness is evident from the text of the scerpts itself.

4 Conclusion

Input-driven control of rule-based expert systems offers both a methodological aid during expert system development and an implementation technique for achieving an appropriate external behavior. The methodological improvement rests upon an additional declarative power given to the rule-based paradigm, and allows expert system developers to concentrate on the conceptualization of the problem domain, while leaving procedural aspects to the procedural reading of the specification of external behavior made by the control strategy. Input-driven control, as implemented in the Milord II shell, supports appropriate interaction with the task environment while retaining correctness of problem solving. Internal and external behavior cooperate in this way for (re)producing correct solutions to problems through a right interaction with the task environment.

Acknowledgement

The core of the research presented in this paper has been carried out at the Institut d'Investigació en Intel·ligència Artificial of the Centre d'Estudis Avanats de Blanes. I am very grateful to the research team at Blanes, and special thanks are debt to Dr. Jaume Agust-Cullell for his support and encouragement.

References

1. C. Barroso. ENS-AI: Un Sistema Experto para la Enseñanza. In *Proc. European Conf. about Information Technology in Education: A Critical Insight*, Barcelona, Nov. 1992.

2. D. C. Berry and D. E. Broadbent. Expert Systems and the Man-Machine Interface. Part Two: The User Interface. *Expert Systems*, 4(1):18–28, Feb. 1987.
3. L. Godo, R. López de Mántaras, C. Sierra, and A. Verdaguer. MILORD: The Architecture and the Management of Linguistically Expressed Uncertainty. *Int. Journal of Intelligent Systems*, 4(4):471–501, 1989.
4. J. A. Hendler, editor. *Expert Systems: The User Interface*. Ablex, Norwood, New Jersey, 1988.
5. J. M. Herod and A. T. Bahill. Ameliorating the "Pregnant Man" Problem. In A. T. Bahill, editor, *Verifying and Validating Personal Computer-Based Expert Systems*, chapter 3, pages 26–45. Prentice Hall, Englewood Cliffs, New Jersey, 1990.
6. J. M. Herod and A. T. Bahill. Ameliorating the Pregnant Man Problem: A Verification Tool for Personal Computer Based Expert Systems. *Int. Journal of Man-Machine Studies*, 35:789–805, 1991.
7. R. López de Mántaras, J. Agustí, E. Plaza, and C. Sierra. MILORD: A Fuzzy Expert Systems Shell. In A. Kandel, editor, *Fuzzy Expert Systems*, chapter 15. CRC Press, Boca Raton, Florida, 1991.
8. J. McDermott and C. Forgy. Production System Conflict Resolution Strategies. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems*, pages 177–199. Academic Press, New York, 1978.
9. J. Puyol, L. Godo, and C. Sierra. A Specialisation Calculus to Improve Expert Systems Communication. In B. Neumann, editor, *Proc. 10th European Conf. on Artificial Intelligence ECAI 92*, pages 144–148, Vienna, Austria, Aug. 1992.
10. J. Puyol, C. Sierra, and J. Agustí-Cullell. Partial Evaluation in Milord II: A Language for Knowledge Engineering. In *Proc. Europ-IA 91*, pages 193–207, 1991.
11. R. Sauers. Controlling Expert Systems. In L. Bolc and M. J. Coombs, editors, *Expert System Applications*, pages 79–197. Springer-Verlag, Berlin, 1988.
12. C. Sierra. *MILORD: Arquitectura Multi-Nivell per a Sistemes Experts en Classificació*. PhD thesis, Universitat Politècnica de Catalunya, May 1989.
13. G. Valiente. Verification of External Adequacy in Rule-Based Expert Systems using Support Graphs. Free session contribution to the *European Workshop on the Verification and Validation of Knowledge-Based Systems EUROVAV 91*, Cambridge, England, July 1991. Published as Research Report 91/15, Centre d'Estudis Avançats de Blanes.
14. G. Valiente. Using Layered Support Graphs for Verifying External Adequacy in Rule-Based Expert Systems. *SIGART Bulletin*, 3(1):20–24, Jan. 1992.
15. G. Valiente. *Heuristic Question Selection*. PhD thesis, Universitat Autònoma de Barcelona, 1993. Forthcoming.

Case-based planning for medical diagnosis

Beatriz López* and Enric Plaza**

Institut d'Investigació en Intel·ligència Artificial (IIIA, CSIC),
Camí de Santa Bàrbara, s/n., Blanes, Girona, Spain.
bea@ceab.es and plaza@ceab.es

Abstract. In this paper we describe a case-based planner (BOLERO) developed for learning the procedure of a diagnosis in a medical expert system. A diagnostic plan is build according to the most recent information known about a patient. BOLERO has been tested in a real application of pneumonia diagnosis. Results show that BOLERO is able to acquire enough strategic knowledge to perform a diagnostic procedure with a high degree of success.

1 Introduction

For some time it has been argued that keeping separate and distinct "control" knowledge from domain knowledge as represented in an expert system is convenient for expert system maintainability and for the reuse of shells on similar problems[2]. In earlier experiments in our Institute, a meta-level architecture was developed with strategic (or control) knowledge represented at the meta-level and domain knowledge being represented at the object level [5]. We also discovered while designing a medical diagnosis expert system[14], that acquiring the strategic knowledge was complicate and time-consuming for the expert and the knowledge engineer. We present in this paper an experiment in the automated acquisition of strategic knowledge for expert systems by means of a case-based approach.

In more concrete terms, strategic knowledge in the expert systems built at the IIIA was to decide upon which diseases were plausible or useful to try to prove (or disprove for dangerous illnesses), and which questions to ask that were relevant to those ends at each moment relative to the known facts. The questions should be made also in a meaningful order and the disease hypothesis should be pursued in a reasonable order, lest the physicians using the system complain about it doing silly things and acting "erratically". In [5] strategic knowledge take the form of plans of action, i.e. a dynamically-changing scheme of the goals worth pursuing. Acquiring this knowledge was slow and based on a trial and error process, until the medical application for diagnosing pneumonia performed according to what was expected. Exploiting the modularity of this metalevel architecture and the separate representation of domain and strategic knowledge, we investigated on the possibility of automatically learning the diagnostic procedure (strategic knowledge) given (a) the domain knowledge, and (b) an expert authenticating the desired system behavior. This is the experiment reported here, where strategic knowledge, represented by

* Supported by a grant from the MEC.
** Partially supported by Project AMP CYCIT 90/801.

plans, are learned using a case-based approach. Plans learned are reused to solve new problems, in such a way that BOLERO acts as the planner of a rule-based system (RBS) with scarce strategic knowledge. The structure of the paper is as follows: in the next section we introduce the metalevel architecture in which BOLERO and a RBS collaborate during problem solving, and then, in section 3 we describe BOLERO, the case-based planner. In section 4 we explain the results of BOLERO when applied to learn and build plans for pneumonia diagnosis. Finally we relate our research to other works, and we give some conclusions.

2 The BOLERO-RBS Reactive System

The BOLERO-RBS system is the result of the integration of the case-based system BOLERO and a rule-based system (RBS) within a metalevel architecture. BOLERO plays the role of the meta-level and the RBS plays the role of the object-level (figure 1). The RBS has knowledge about a specific domain in order to solve a problem while BOLERO has planning knowledge particular to that domain. The BOLERO-RBS system is a reflective system [11]: the meta-level (BOLERO) is able to modify the state of the object-level (RBS) through plans, and it is also in charge of starting and stopping the system execution.

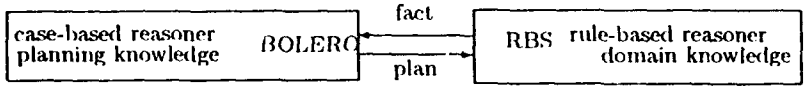


Fig. 1. The BOLERO-RBS system.

For the sake of simplicity we will not go on details about how both systems interact (this has been the aim of the paper [10]). Essentially BOLERO builds plans according to the state of the RBS when solving a problem and plans are executed by the RBS. One action in BOLERO means in the RBS to validate a goal by chaining rules. When the RBS is validating a goal an action is being executed, and when a goal has been achieved, the action is completed. The complete execution of an action leads to know some evidence about patient illness. For example, the action *mycoplasma* is interpreted by the RBS as the goal *pneumonia-caused-by-mycoplasma*; when it is known that it is *quite-possible* that the *pneumonia is caused by mycoplasma* then the action is complete and some evidence about the patient illness is found. When the RBS obtains new information as a consequence of the execution of a plan, BOLERO can generate a new plan adequate to the new circumstances. Thus, the validation of any goal in the RBS can be interrupted, if BOLERO dynamically generates a new plan. In the same way, when an action a is interrupted in a given moment of the problem solving process, the action can be continued in a later moment if BOLERO builds a plan that contains a. To continue an action means to go on with the validation of the action's goal in the RBS. The generation and execution of plans is then interleaved in BOLERO-RBS, in such a way that the integrated system behaves as a reactive planner [4].

Reactive planning has been demonstrated as an useful technique to deal with uncertainty and incompleteness, a constant feature in medical applications[10]. The information in medical diagnosis is incomplete because many situations cannot be fully represented since there is no way of gathering all the information needed and at the same time the information available is uncertain because lacks precision or requires subjective assessments. For example, let us suppose that a 25 year old patient with a risk factor of infection by HIV arrives at the hospital suffering from pneumonia. The physician plans a diagnostic procedure in order to find the agent causing the pneumonia. This procedure or plan consists of the following actions: take an X-ray of the thorax, do an hemography, and a gram-sputum test. Let us assume that the physician sees an interstitial pneumonia in the X-ray. With this information and taking into account the risk factor by infection by HIV, the physician considers the possibility that the patient has AIDS. Consequently he decides to give up the rest of the pending actions (i.e. hemography and gram-sputum test) and to realize some new action (like for example a broncho-alveolar enema), to determine whether Pneumocistii carinii is causing the pneumonia. Given the initial information available about a patient the physician then generates a plan p, and when his knowledge about the patient is enriched as a consequence of starting the execution of p, he produces a new different plan p'. BOLERO, as a reactive planner takes into account the incomplete information problem in such a way that it can generate plans dynamically, anytime some new information about a problem is available to the system.

3 The BOLERO System

In this section we explain the knowledge represented in BOLERO (section 3.1) and the components of BOLERO as a case-based planner: the learning method (section 3.2), how BOLERO builds plans (section 3.3), and the plan evaluation method (section 3.4.

3.1 Knowledge Representation

Planning knowledge in BOLERO is based in the information stored in the cases provided by the teacher. Cases determine the plan memory from which BOLERO is able to solve problems.

Cases. A case is the set of episodes observed when a teacher solves a problem(figure 2): $C^i = [\delta_0^i, \delta_1^i, \ldots, \delta_{n_i}^i]$. Each episode is a pair $\langle s_j^i, p_j^i \rangle$ formed by a situation s_j^i and a plan p_j^i. In the first episode, s_0^i represents the initial state of a problem, and p_0^i a plan containing the immediate action a_0^i to start problem solving. The situation of the next episode, s_1^i, represents the state of the problem achieved after that a_0^i has been applied, and p_1^i represents the concatenation of the plan p_0^i and a new action a_1^i with which to continue problem solving given the new information in s_1^i. And so on until the last episode, that contains the final state $s_{n^i}^i$ and the plan $p_{n^i}^i$ that leads from s_0^i to $s_{n^i}^i$. Then $p_{n^i}^i$ is the solution of the problem from the point of view of planning. The plan $p_{n^i}^i$ is called the final plan meanwhile the rest of the plans of a case are called partial plans. Each state or situation s_j^i of a case is composed by a set of facts. A fact f_i is a pair (id_i, v_i) where id_i is the identifier of the

fact, and v_i the value. The final state $s^i_{n^i}$ must contain the diagnosis of the patient illness or domain solution $S^i_{C^i}$. For example the domain solution of the case represented in figure 2, S_{g22v1}, is: { *(pneumococcal-pneumonia, possible), (pneumonia-by-enterobacteriaceae, moderately-possible), (pneumonia-by-legionella, quite-possible), (tuberculosis, very-little-possible), (pneumonia-by-pseudomonas, unknown)*}. The domain solution comes from the result of individual actions $a^i_j \in p^i_{n^i}$, that we note by $result(a^i_j)$. For example, *result(pneumococ)* = { *(pneumococcal-pneumonia, possible)*}.

δ_0: s_0 = { }
p_0 = [gather-information]
δ_1: s_1 = { (community-acquired, *yes*), (state-of-patient, *moderately-serious*), (stablishment, *sudden*), (antecedents, *immunodefficiency*), (kind-of-inmunodefficiency, *advanced-cancer, treatment-with-antineoplastic-agents*), (X-rays, *pleural-effusion*), (dispnea, *no*), (sample-of-pleural-liquid, *yes*)
p_1 = [gather-information, effusion]
δ_2: s_2 = s_1 ∪ { (exudate-effusion, *unkown*), (pleural-stain-made, *yes*), (germs-pleural-stain, *yes*), (empyema, *certain*), (germ-identification, *unknown*), (*gram-stain, grampositive-coccus-in-pairs*), ... }
p_2 = [gather-information, effusion, bact-atip]
δ_3: s_3 = s_2 ∪ { (bacterian-pneumonia, *certain*), (headache, *unknown*), (atipical-pneumonia, *unkown*) }
p_3 = [gather-information, effusion, bact-atip, pneumococ]
...
δ_8: s_8 = s_7 ∪ { (pneumonia-by-pseudomonas, *unkown*) }
p_8 = [gather-information, effusion, bact-atip, pneumococ, *enterobacteria, legionella, tuberculosis*, pseudomonas, □].

Fig. 2. *Some partial plans and the final plan of case "g22v1".*

Plan Memory. The plan memory is organized according to the episodes of cases. Since every initial episode δ^i_0 of a case have the same start plan (i.e. $\delta^i_0 = \langle s^i_0, p\rangle$),then it is possible to develop an hierarchy organization of cases. A node N^i is a tuple $\langle v^i, p^i, \tau^i\rangle$ made up by a *generalized situation* v^i, a partial plan p^i, and the typicality τ^i of p^i. The generalized situation v^i is the result of a generalization process of all the situations s of the episodes δ^k_j that contains the plan p^i (see next section). Generalized situations v^i are composed by generalizations, i.e., $v^i = \{g^i_j\}$. Each generalization g^i_j is a tuple $\langle id^i_j, v^i_j, w^i_j, \sigma^i_j\rangle$ where id^i_j is the identifier, v^i_j a value or set of values, w^i_j the strategic relevance of the generalization, and σ^i_j the frequency. The strategic relevance is a number defined in the unit interval and points the strength between a generalization g^i_j and a plan p^i. A generalization g^i_j with a strategic relevance 0 means that the generalization has been completely irrelevant respect when building the plan p^i of the node N^i. A generalization with an strategic relevance 1 is definitely relevant. Identifiers of facts are used in BOLERO to index generalizations with the same identifier, and the strategic relevance of the generalization plays an important role when recovering cases from memory. The strategic relevance is in some way the substitute of the frequency on other learning mechanisms as COWEB [3] and LAMDA [12]. The indexing mechanisms of such system heavily rely on the frequency of facts but, particularly in medical domains, we detected that a fact with a relative low frequency σ might be very relevant. Specific data tends to be more relevant, although they are not often available.

The plan of a node N^i has one action more than the plan of its direct ancestor

N^{i-1} and one action less that the plan of its direct successors N^j, in such a way that $p^j = p^i; a_k$ where, obviously, a_k is different for each successor of N^i. The typicality τ^i is the number of times that p^i has been applied in the cases. A *leaf node* N_t^i in addition to the information concerning to a node, has a set of solutions S^i, $N_t^i = (v^i, p^i, \tau^i, S^i)$. The set of solutions S^i contains the domain solutions of the cases that have p^i as the final plan: $S^i = \bigcup_j S_{C^j}$.

3.2 Learning Plans

BOLERO learns plans by organizing in its memory the cases, $C^0, C^1, \ldots, C^m$ of a training set. BOLERO learn plans incrementally: first of all the memory is empty and cases are incorporated one by one. The incorporation of a case in memory is based on two main methods: a generalization method and a termination conditions learning method.

Incorporation of a Case. A case C^i is added to the memory by comparing the plan p_j^i of its episode δ_j^i with any node of the level j of the hierarchy. If there is a node N^k in level j that has a plan $p^k = p_j^i$, then the generalized situation v^k of N^k is updated (see next paragraph). The incorporation of the case follows with the next episode δ_{j+1}^i of the case C^i and the nodes of the level $j+1$ successors of N^k. The *incorporation method* is applied in each episode of the case. If there is no node with a partial plan like p_j^i, then a new branch of the hierarchy is started from level i-1 and the rest of the episodes of the case, $\delta_j^i, \ldots, \delta_{n^i}^i$, become nodes of the new branch. The last episode of a case is stored in a leaf node where termination conditions learning takes place.

Generalization. When a plan has been applied to two different situations (i.e. there are two different episodes made up by the same plan and different situations) the generalization method provides a new generalized situation that covers the two original situations. Given a generalized situation v^i of a node and an episode δ_j^k of a case, the generalization procedure consist in the following steps:

1. Perform a generalization of the values of the facts f_1 in s_j^k and generalizations g_2^i in v^i with the same identifier. The generalization of two values v_1 and v_2 is the union of the two values $\{v_1\} \cup \{v_2\}$.
2. If an abstraction between a fact a generalization with different identifiers is possible, do it. Abstraction of facts is based on relations defined among the identifiers of facts (or generalizations) in the domain network (see figure 3). Given a set of fact identifiers id_1, id_2, id_n , abstraction produces a new identifier id. The new identifier id has a relation with each id_i.
3. Otherwise, add the fact to the generalized situation v^i of the node.

Termination Conditions Learning. One particularity that presents planning in a medical domain is that the goal specification is unknown prior the elaboration of a plan. That is, planning has been classically formulated as the process of building a sequence of operations that achieves a goal state from an initial state. Both the goal and the initial state are given as the problem statement. In medical diagnosis the initial state is given but the goal state is unknown. To know the goal state is equivalent to know the solution of the problem, i.e. the patient illness. If the goal state is not provided, BOLERO must have some method to recognize when

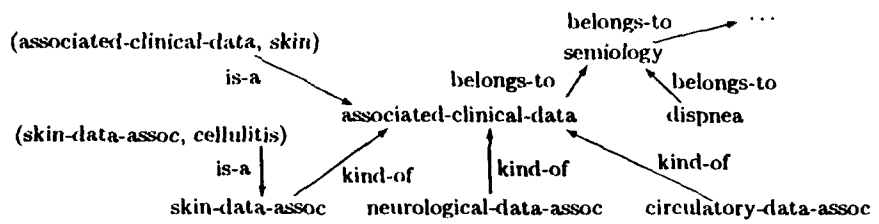


Fig. 3. *Partial view of the domain knowledge network where different relations among types of facts are defined (belongs-to, kind-of, etc.). Facts are connected with a is-a link to the identifier*

an adequate solution for a problem is reached. Termination condition learning occurs in BOLERO in two different ways: 1) by comparison with the domain solution provided in the cases, and 2) by domain knowledge. Solutions S^j stored in leaf nodes N_i^i can be reminded when solving new problems, and if a similar solution is found then problem solving of the new problem is terminated. However S^i are weak conditions in the sense that it is very rare that two cases have the same domain solution, even they have the same final plan. For example, the cases "r13v1" and "m07v1" have the same final plan */gather-information, bact-atip, pneumococ, legionella, enterobacteria, s-pyogenes/*, but their domain solutions are different because the facts values gathered during the execution of the actions have been different. So while the solution of case "r13v1" is { *(pneumoccocal-pneumonia, certain), (pneumonia-by-legionella, very-little-poss), (pneumonia-by-enterobacteriaceae, very-little-possible), (pneum-by-s-pyogenes, unknown)*}, the solution of the case "m07v1" is { *(pneumoccocal-pneumonia, possible), (pneumonia-by-legionella, very-possible), (pneumonia-by-enterobacteriaceae, possible), (pneum-by-s-pyogenes, unknown)*}. For this reason BOLERO also uses a set S_c of strong conditions acquired from a domain expert. That is, if any condition in S_c is satisfied, we say that the domain solution of a problem is achieved, and so problem solving has to be stopped.

3.3 Plan generation

To solve a problem π^i means building a plan that achieves a domain solution (in a situation $s^i_{n^i}$) given the initial situation s^i_0. While solving a problem, different plans are build according to the different situations achieved as a consequence of interleaving generation and execution of plans. Current situation s^i_j is reached through the application of a plan p^i_{j-1} in the previous situation s^i_{j-1}. The execution of p^i_{j-1} leads to the knowledge of a new fact f that determines the new situation $s^i_j = s^i_{j-1} \cup \{f\}$. In the new situation s^i_j BOLERO builds a plan p^i_j, eventually different from the plan p^i_{j-1} and more adequate to the new circumstances. The method to build a plan (as any other case-based system) is based in two basic steps: retrieval and adaptation. Both steps are performed each time that a new situation s^i_j is known, until BOLERO decides to terminate the problem solving process.

Plan Retrieval. Let us assume we are solving the problem π^i, and that we know a set of facts or current situation s_c^i. The retrieval method consists on reminding generalized situations similar to s_c^i. Reminding occurs in a three step procedure: (1) first index the memory by the identifiers of the facts of s_c^i and, using and a spreading activation mechanism[7], recover generalized situations $\mathcal{G} = \{\upsilon^j\}$; (2) compute the similarity between facts f_a in s_c^i and generalizations g_k^j in each υ^j using the function m_f; and finally (3) compute the overall similarity between s_c^i and each situation υ^j in G using the aggregation function m_s. Given a generalization g_k in memory and a fact f_a in s_c^i with the same identifier (otherwise $m_f(g_k, f_a) = 0$),

$$\mathrm{m_f(g_k, f_a)} = \begin{cases} 0 & if \ \ monovalued(f_a) \ \ \wedge \ \ v_a \neq v_k \\ C_n(W_k) & if \ \ multivalued(f_a) \ \ \wedge \ \ v_a = v_k \\ w_k(1 - d(v_a, v_k)) & if fuzzy(f_a) \\ w_k & otherwise \end{cases}$$

where C_n is a disjunctive aggregation function of n evidences based on a C-norm[12, 1]. The function m_s combines the evaluation m_f for each generalization g_k^j in υ^j, in the following way:

$$m_s(\upsilon^j, s_c^i) = \frac{\frac{\sum_{g_k^j \in \upsilon^j} m_f(g_k^j, f_a)}{n+\lambda} + \frac{\sum_{g_k^j \in \upsilon^j} m_f(g_k^j, f_a)}{|s_c^i|}}{2}$$

where n is the number of generalizations in υ^j that have frequency 1.0; λ is the number of facts in υ^j that are also in s_c^i and the frequency of which is less than 1; and $|s_c^i|$ is the cardinality of s_c^i. The m_s function computes a mean among the inclusion (first term of the numerator) and the exclusion (second term) of the facts of the current situation in a generalized situation.

Plan Adaptation. The adaptation of the plan p^k of the node N^k retrieved from memory consists on avoiding the repetition of actions already executed in the current situation s_c^i. That is, if an action a_j has been already performed in a given moment x, assuming that situations are monotonic, for any later situation s_y^i ($s_x^i \prec s_y^i$), $result(a_j^i) \subset s_y^i$. So, there is no sense to include again the action a_j in a plan to continue problem solving. Then, the outcome of the adaptation of the plan p^k retrieved from memory, is the new plan p_c^i where any action a_l of p_c^i belongs to p^k but $result(a_l) \not\subset s_c^i$. For example, let us suppose that in a given situation s_j^i the plan p = */gather-information, bact-atip, pneumococ, enterobact, legionella/* has been applied and the result of all its actions are known. Let us also suppose that with in the situation s_j^i a new node N^k is retrieved, whose plan is p^k = */gather-information, bact-atip, estafilococ, pneumococ, enterobact, tuberculosis/*. Then the plan generated is p_c^i = */estafilococ, enterobact, tuberculosis/*. Actions *gather-information*, *bact-atip*, and *pneumococ* of p^k are not included in p_c^i since they have been already completed.

Termination. There exists four possibilities to end the problem solving: (1) when there is no active node retrieved from memory; (2) when some termination condition described in S_e is satisfied in the current situation s_c^i; (3) when the current domain solution and a solution in a leaf node (i.e. a solution of a previous case) are similar; and (4) when there is a final plan of a node N_l^i already executed in the current situation s_a and the computed overall similarity for the current retrieved node N^j is $m_s(\upsilon^j, s_a) = \alpha \ll \beta = m_s(\upsilon^i, s_a)$ [9].

3.4 Plan Evaluation

Plan evaluation in medical diagnosis provides information about the achievement of the diagnostic of the patient, (the *correctness* of the plan), and about the quality of the procedure followed to achieve the diagnostic (the *accuracy* of plans). Evaluation requires some kind of knowledge that we call *gold standard*, i.e. the right solution, the wished solution. The gold standard is a model of functionality, that for plan evaluation purposes is the optimal plan. To determine a gold standard it is not an easy task as has been demonstrated elsewhere [8]. For this reason we have defined in BOLERO an approach to the gold standard that we call the evaluation standard (ES). The ES is defined for each training case C and consists on an admissible plan $PA_p(C)$ and a set of domain solutions $PA_d(C)$. On one hand, PA_p is provided by a oracle. On the other hand, PA_d is an approach to the ideal domain solution that we build upon the consensus of several experts. BOLERO uses the ES to determine the correctness and the accuracy of plans.

Correctness of Plans. After solving a problem, BOLERO obtains a new case C^i. The plan $p^i_{n^i}$ succeeds (and so the plan is correct) if all the actions required by PA_p are in $p^i_{n^i}$. That condition of success of plans guarantees that at least BOLERO has found the same diagnosis than the teacher, since BOLERO has performed the same actions with the same data. It is possible that some actions performed by the teacher do not lead to know the diagnosis of the patient. However such actions should not be forgotten since in medical diagnosis it is preferable to perform additional actions than to forget or discard actions that may lead to diagnose some dangerous disease. Thus we say that BOLERO has a conservative bias. The degree of success of a plan $p^i_{n^i}$ relates the number of actions of the PA_p included in $p^i_{n^i}$ and the total number of actions of PA_p. So, a plan is correct if the degree of success is 100%.

Accuracy of Plans. Besides the degree of success of plans we have defined the *degree of focus*. The degree of focus is easily defined as the contrary of the *degree of out-focus*: degree of focus = 100 - degree of out-focus. The degree of out-focus is the percentage of actions performed that could be spared. Among all actions of a final plan that are not in PA_p, we consider a unnecessary action the action the results of which are not justified by the domain experts (i.e. those not in PA_d).

To illustrate with an example plan evaluation, let us suppose that BOLERO has solve the case "g24v1" with the plan p_{g24v1} = *[gather-information, bact-atip, virus, chlamydia, q-fever]*, and that the PA_p is *[gather-information, bact-atip, virus, mycoplasma, chlamydia]*. The action *mycoplasma* is not included in p_{g24v1}, and then the plan build has a degree of success of 80%. The degree of focus is 100% because, although the action *q-fever* is not in PA_p("$g24v1$"), it is contained in PA_d("$g24v1$").

4 An Application to Medical Diagnosis

We have applied BOLERO to build plans to diagnose the agent causing pneumonia. Pneumoniae are frequent illnesses that need urgent treatment[14]. An early treatment means that the physician needs to diagnose the agent that causes the infection by taking advantage of all the information he has about the patient, and before knowing the results of some tests. Although complementary proves can be performed, the

physician does not know generally the etiology of the illness. Incomplete and uncertain information are characteristics of pneumonia diagnosis that persist along all the therapeutical process. Decisions should be taken and the experience of the physician on previous cases plays an important role in the diagnosis of the patient.

Our start point has been a set of 79 cases provided by a teacher (a physician expert in diagnosing pneumoniae) and correspond to real cases taken from 4 hospitals. We have a set of 460 facts representing the pneumonia domain. Each case has in average 88,23% of facts, and 22,87% of fact's values are unknown. To execute the plans build by BOLERO, we have a rule-based system available, PNEUMON-IA [14] that is able to interpret and execute the plans.

We have evaluated experimentally BOLERO by using the leave-one-out method. An experiment consist in training the system with 78 cases and testing the system with a test case C (selected randomly among the 79 cases). We have performed 10 experiments, with 10 different test cases, and changing the order of the cases in the training set. To evaluate BOLERO we have defined two measures: identification and predictiveness. Identification consist on evaluating the degree of confidence that the system shows in solving a case already in memory (i.e. a case of the training set). Predictiveness determines the degree in which BOLERO is able to solve a case not seen before (i.e. a test case). The measurements have been taken when BOLERO has n cases in memory, then $n+k$, then $n+2k$, and so on until all cases have been learned.

Results on identification show that BOLERO learns to solve the cases exactly (100%) as the teacher does. The number of cases in memory does not affect the identification of a case in memory. Results on predictiveness are given in figure 4. The behavior of the system is really good. Just with few cases in memory BOLERO is able to solve problem around the 98% of degree of success and the 80% of degree of focus. A full experimental evaluation of the system is provided in [9].

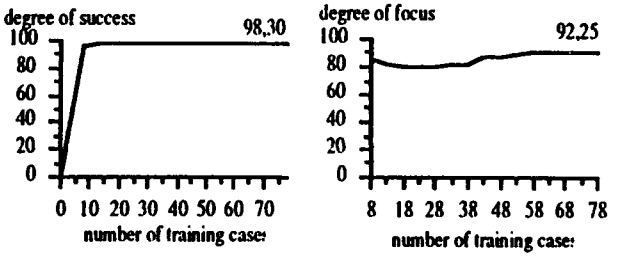


Fig. 4. *Predictive results on BOLERO.*

5 Conclusions

In the experiment showed in this paper, case-based planning has been proved as an useful technique to acquire strategic knowledge. Degree of success is achieved after the training of a few cases, and accuracy increases with the number of training cases. The main conclusion is that strategic knowledge can be acquired in few minutes

automatically from the medical cases and the expert advice, while hand-coding the strategic knowledge the expert or a knowledge engineer spends many days, two weeks fulltime in the pneumonia example shown in the paper [14]. Moreover, the medical cases needed in order to learn plans were also needed in the hand-coding process to validate the strategic knowledge, so no new effort was required.

Two main case-based planners are related to our research: CHEF [6] and SMART [13]. BOLERO has an innovative approach to case-based planning fostered by the fact that BOLERO has been designed to be useful for medical diagnosis. The evaluation method, the termination condition learning method, and the capability of performing a reactive planning are characteristics of the systems that neither CHEF nor SMART have.

References

1. P.P. Bonissone and K.S. Decker. Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-off Precision and Complexity. In L.N. Kanal and J.F. Lemmer, ed., *Uncertainty in AI*, p. 217–247. Elseview Science Pub., North-Holland, 1986.
2. W.J. Clancey. The Advantages of Abstract Control Knowledge in Expert Systems. In *Proc. AAAI*, pages 74–78, Washington, D.C., 1983.
3. J.H. Genenari, P. Langley, and D. Fisher. Models of Incremental Concept Formation. In J. Carbonell, editor, *Machine Learning. Paradigms and Methods*, pages 11–61. MIT/Elsevier, 1990.
4. M.P. Georgeff and A.L. Lansky. Reactive Reasoning and Planning. In J. Allen and A. Tate, editors, *Readings in Planning*, p. 729–734. Morgan Kaufmann Pub. Inc., 1990.
5. L. Godo, R. López de Màntaras, C. Sierra, and A. Verdaguer. MILORD: The Architecture and Manegement of Linguistically Expressed Uncertainty. *International Journal of Intelligent Systems*, 4(4):471–501, 1989.
6. K.J. Hammond. *Case-Based Planning. Viewing Planning as a Memory Task*, volume 1 of *Perspectives in Artificial Intelligence*. Academic Press, Inc., 1989.
7. J.A. Hendler. The Design and Implementation of Marker-passing Systems. *Connection Science*, 1(1), 1989.
8. B. López. CONKRET: A Control Knowledge Refinement Tool. In M. Ayel and J.P. Laurent, editors, *Validation, Verification and Test of Knowledge-Based Systems*, chapter 13, pages 191–203. John Wiley & Sons, 1991.
9. B. López. *Aprenentatge de plans per a sistemes experts*. PhD thesis, Universitat Politècnica de Catalunya, Facultatd d'Informàtica de Barcelona, 1993. In preparation.
10. B. López. Reactive Planning Through the Integration of a Case-Based System and a Rule-based System. In *Proc. AISB*, University of Birmingham, UK, 1993. To appear.
11. P. Maes. Issues in Computational Reflection. In P. Maes and D. Nardi, editors, *Meta-level Architectures and Reflection*, pages 21–35. Elsevier Science Pub. B. V., 1988.
12. N. Piera, Ph. Desroches, and J. Aguilar-Martin. LAMDA: An Incremental Conceptual Clustering Method. Technical Report 89420, Laboratoire d'Automatique et d'Analyse des Systèmes (LAAS), Toulouse, France, 1989.
13. M.M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992.
14. A. Verdaguer. *PNEUMON-IA: Desenvolupament i Validació d'un Sistema Expert d'Ajuda al Diagnòstic Mèdic*. PhD thesis, Universitat Autònoma de Barcelona,1989.

MethoDex :
A Methodology for Expert Systems Development

J.P.Klut and J.H.P. Eloff
Department of Computer Science
Rand Afrikaans University
PO Box 524
Johannesburg
2000
South Africa

Abstract:
A framework that can help users, experts and data processing personnel to be effectively involved in the development of Expert Systems is proposed and discussed. The approach taken is to evaluate the two most basic approaches used in industry today, namely, the usage of a standard methodology (SDLC) which is also used for the development of general business systems, and the well-known Knowledge Engineering Cycle approach. The proposed methodology for Expert Systems development (MethoDex) is designed to consist of a three-layered framework: activities, procedures and resources. MethoDex has been developed within a life assurance industry environment (Liberty Life Association of Africa Limited) and has already proved itself as a valuable framework for the successful development of Expert Systems within a commercial environment.

1 Introduction

It is a fact that today Expert Systems (ESs) is a fast growing and developing information technology. The programming of knowledge to create commercial applications has been made feasible with the introduction of many types of knowledge representation tools, of which the "Shell" is probably the best known. A major reason for the acceptance of this technology can be attributed to the relative ease of use that the early commercial "Shells" exhibited and their non-procedural way of representing knowledge, mostly in terms of rules. Examples are VpExpert, Level5, Exsys and Personal Consultant Plus.

Early identified knowledge representation schemes (such as rules) proved to be not functional for all types of knowledge problems [1]. This fact, combined with the realization that the building of ESs is moving out of the AI Labs into the normal corporate systems development areas, [2] have led to systems development and management problems [3].

There are many references to the development and management of ESs, but they are either specific to a particular Expert System (ES) application [4] or simply address the Knowledge Engineering design and development life cycle [5]. In most

cases these references include the advocating of prototyping [6] and in many others a strong procedural approach [7] is adopted. However, very little reference to a general systems development life cycle of an "Expert System", or knowledge-based application, exists. This fact highlights the current nature of the project management of ESs and adds to the confusion and difficulty in the practical implementation thereof. Without a clear view and guidelines of how to manage and develop ESs, this predicament will continue, much to the detriment of the acceptance and growth of this technology.

The building of an ES has many facets, of which the engineering of knowledge is one. In principle the KE Cycle model [5] is used as a guideline to the Knowledge Engineering process (the analysis, design, development and testing of knowledge or expertise). Knowledge Engineering can be done in many ways and is in many instances dependent on the complexity of the expertise to be represented. To represent expertise, the KE Cycle can be used successfully. In this instance the Knowledge Engineering process is based specifically on a prototyping process model as opposed to a function or data process model [8].

However, to base the overall methodology for development of an ES, from ES inception to ES maintenance, on the KE cycle or KE cycle in combination with the above-mentioned process models, is incomplete. It does not provide a comprehensive solution to the problems and complexities involved in the development of ESs. The first main reason for this is that certain specific activities that mark the formal initiation and finalization of a project are disregarded. The second main reason is that the KE cycle model assumes abstraction as the project development progresses. It is applied in the same way on the macro and micro level for activities and tasks. For example, Analysis (Identification and Conceptualization in the KE cycle) can imply analysis of:

a) a single expertise entity (a fact or object),
b) knowledge (eg. a rule) or even knowledge base structuring, or
c) the total project from a management and Knowledge Engineering point of view.

Procedures in this approach thus do not refer to a series of specific steps and can over emphasize the abstract concepts inherent in the KE cycle.

The use of a conventional Systems Development Life Cycle (SDLC) approach for the development of ESs, also has shortcomings. The main shortcoming is that in a conventional SDLC the underlying basis is the programming of procedures; in an ES the underlying basis is the programming of knowledge.

This situation account for many ES project management problems. Management has to manage an ES project using a combination of the above approaches. This is the basis for the methodology *MethoDex* as proposed later in this paper.

2 DESIRED COMPONENTS IN THE EXPERT SYSTEMS DEVELOPMENT LIFE CYCLE.

Based on the Knowledge Engineering model and using a bottom up approach, the authors identified the following primitives needed in the ESs development life cycle:

1. The KE cycle, which is based on a prototyping process model, used to produce

a prototype .

2. Two development deliverables: the demonstration and production ES prototypes.
   The demonstration prototype serves as an analysis, demonstration and risk assessment vehicle.
   The operational prototype is the production version of the demonstration prototype. It is based on the same process model but with a different emphasis on certain procedures.
3. Procedural project management aspects, for example: initial actions taken to identify the ES application and to initiate the project as well as actions taken to ensure the implementation and continuous tuning and updating of the ES application.

To provide a framework to logically incorporate these primitives needed in an ESs methodology, the authors use some principles of the waterfall model [9] as well as the "prototype and risk assessment concept" used in the Spiral model as proposed by Boehm [10]. These principles are used to address the development life cycle components from initiation to completion.

3 MethoDex : A PROPOSED METHODOLOGY FOR THE DEVELOPMENT OF EXPERT SYSTEMS.

The design and assembly of the above desired components are built into a proposed methodology, *MethoDex*, an acronym for "A Methodology for Expert Systems". A high level view of *MethoDex* is shown in figure 1.

A conical layered approach is used to describe three important facets of *MethoDex*:

- The central triangular view of the model shows *MethoDex* activities and their dependencies on one another.
- The right-hand triangular view shows the activities in relation to the conventional systems development life cycle phases. This illustration indicated by the checkered circle on the diagram, shows that in the *MethoDex* case, some of these phases overlap and will be executed in parallel.
- The left triangular view of the model shows some project management components that need to be taken into account during the creation of the ES. In particular, second-level entry points to *MethoDex* refer to those stages in the methodology where sub-knowledge bases or totally new ESs need to be developed as a result of analysis and design criteria. When these entry points are used, *MethoDex* is executed again to address the sub-level of the main ES development effort. *MethoDex* is thus used in the same fashion (possibly with a different emphasis on some *MethoDex* components) for sub-level knowledge definition and creation.

A more detailed view of *MethoDex* is shown in figures 2a and 2b.

4 THE MECHANICS OF MethoDex.

4.1 Activities

Activities in *MethoDex* are logically based on the primitives that are needed in a methodology for ESs development.

Activities serve as milestones or checkpoints and carry important information that is needed in the subsequent activity.

Some activities produce specific deliverables that serve as reference to the management of the ES development (fig. 1, activity A4 and A7). Figures 2a and 2b show how activities are linked horizontally to one another. Linked from left to right and executed only once, this conventional SDLC approach ensure that the development of the ES does not end up in the ES prototyping trap of never ending prototyping [11]. In the authors' opinion this is one of the very real dangers that exist in early ES development. The "one-off execution" feature ensures that the system scope is more easily adhered to and clearly defined. It also simplifies the estimation of project duration. The two most important activities in *MethoDex* are the Demonstration (A4) and Operational (A7) prototype construction.

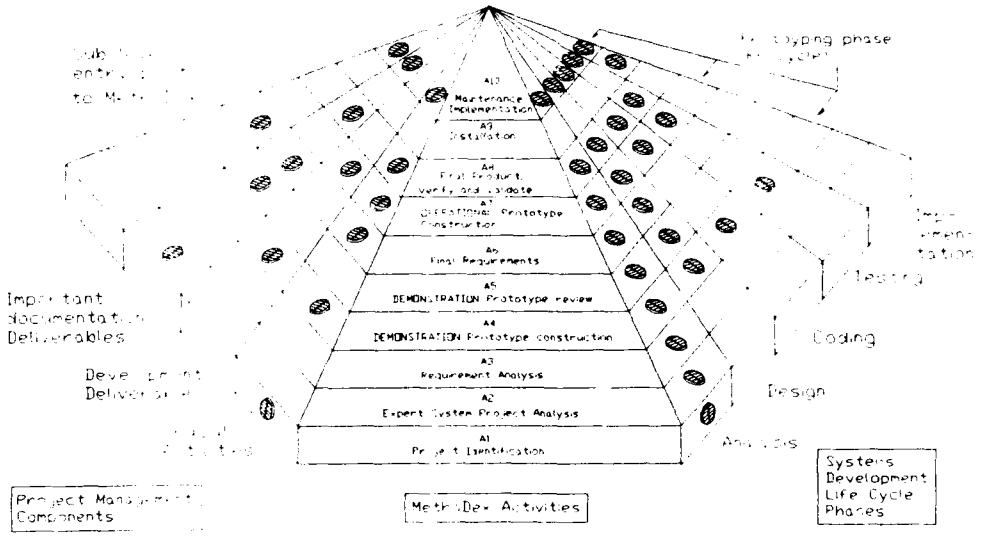


Figure 1
A Conceptual layered high level view of MethoDex

4.2 Procedures

Procedures are a set of guidelines and steps that need to be performed in a certain way to create an environment in which all activities can be successfully completed (see figure 2a and 2b). In the creation of this environment, information in terms of resources and deliverables is used and created.

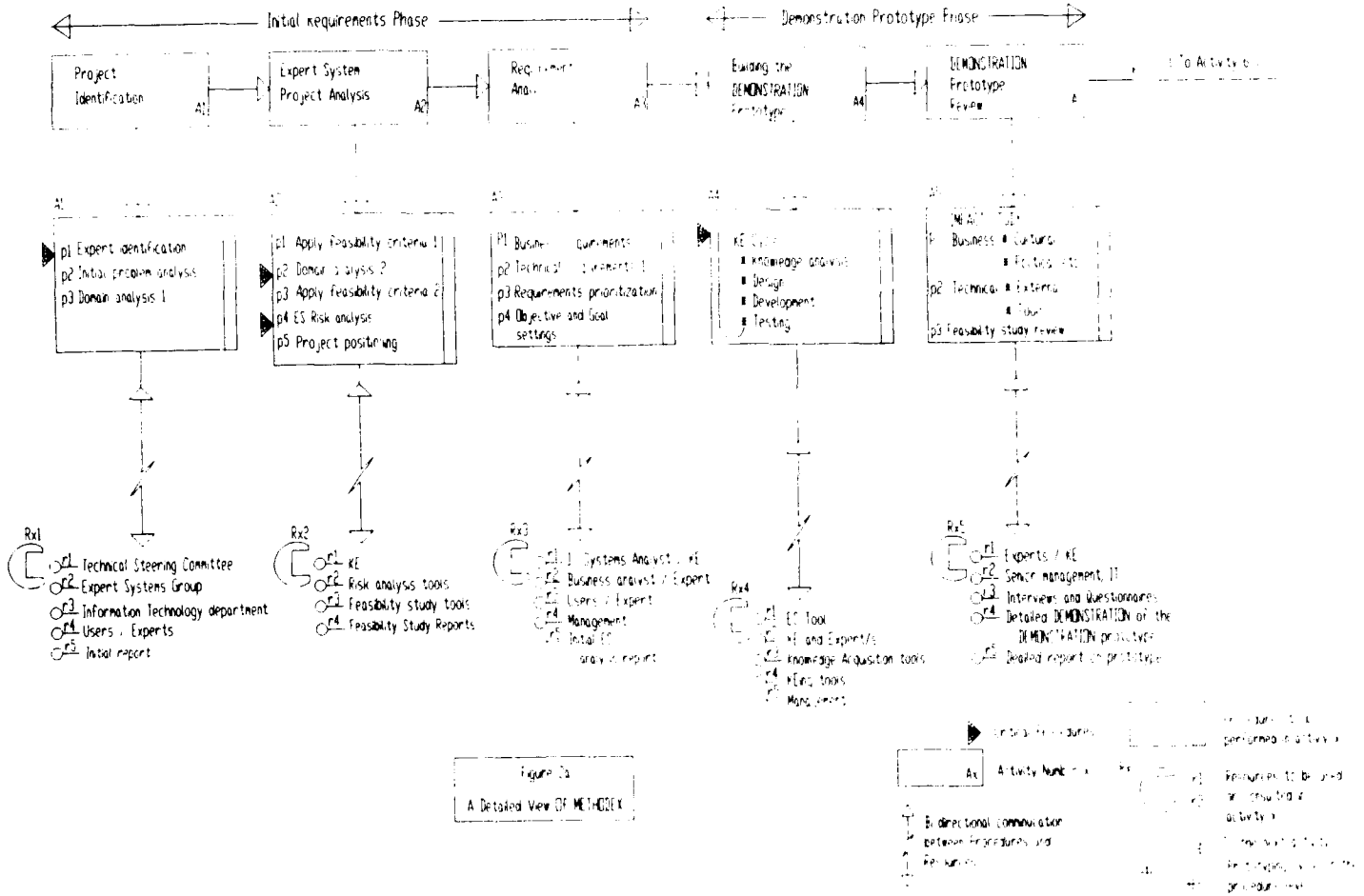


Figure 3a

A Detailed View OF METHODEX

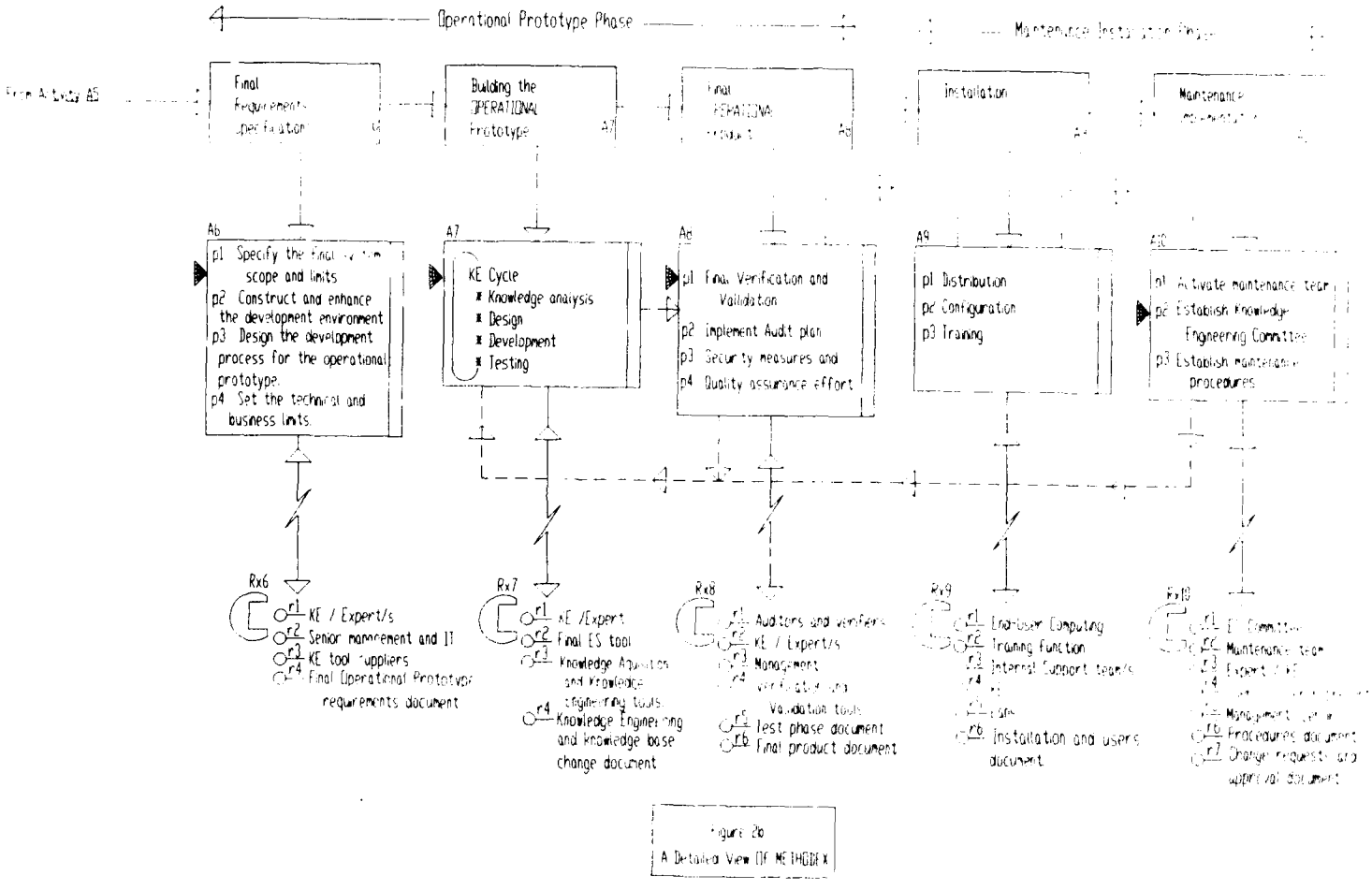


Figure 2b
A Detailed View Of METHODEX

A set of procedures relates to a specific activity. Input or information is supplied from the procedures of previously completed activities. If an activity cannot be completed because of a shortage of information input from preceding activities, neighboring procedures can be revisited to obtain the required input. It is important to note that the development process never moves back to previous activities. When revisited procedures produce a change in information in their activities, this information change may have an influence on the next activity's procedure execution. A good example of this situation is found in activities A7, A8 and A10.

When the Maintenance team is established a knowledge change request needs to be implemented, the procedures of activities A7 and A8 are again executed. Although the activities for these procedures are termed "Building the Operational Prototype" and "Final Operational Product", the current project development activity is known as "Maintenance and Implementation". The reason for taking this approach is to ensure a controlled and structured development effort, which makes functional project management possible.

Procedures can result in deliverables such as the feasibility study(A2,p2&p3).

4.3 Resources

Resources are those components in the methodology that are being used by procedures in an activity to allow for successful completion of the activity. These resource components form part of the domain in which *MethoDex* operates. Resources are classified into four basic categories namely People, Tools, Techniques and Deliverables.

5 THE COMPONENTS OF THE MethoDex METHODOLOGY

Due to the limitation in paper size to accommodate proceeding standards the discussion on the *MethoDex* components are narrowed down to activities A4 and A5. As *MethoDex* is a methodology framework it is suggested that organizations who adopt this as an ES methodology should customize it accordingly, viewing the framework from an activity point of view. The discussion to follow can serve as an example.

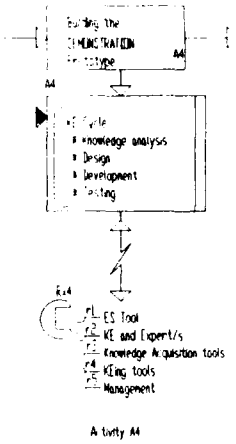
ACTIVITY A4 : BUILDING THE DEMONSTRATION PROTOTYPE

A4 is a crucial activity in the methodology. Any of the succeeding activities are very much dependent on the completion of this activity.

The goals of the demonstration prototype are:

- To serve as a vehicle for gathering the expertise quickly and feasibly [12]. This can be achieved by the use of a knowledge engineering tool (eg. an ES shell) which exhibits feasible prototyping (KE Cycle) capabilities. The use of such a tool maintains the momentum of the analysis and provides an interactive development environment.
- The demonstration prototype must also serve as a model of the proposed system in determining the scope and function of the total project.

- The demonstration prototype serves as a model of the expertise that will be represented or programmed. The model thus assists the KE in determining the overall detailed feasibility and applicability of the proposed expert system.
- The demonstration system also gives the KE an indication of what type of knowledge representation scheme and tool should be used, for example: rules, objects, frames, semantic or neural nets [13].
- The demonstration prototype is an important tool to introduce the experts to the technology, and serves to demonstrate the extent and possibilities of this type of system.



Procedures

Because of the above specific characteristics of the demonstration prototype, the KE should focus on a sub-section of the expertise that is neither too difficult nor too easy to represent [5]. This activity should have a definite and realistic time frame during which specific objectives and issues can be achieved and assessed. Some iterations of the KE Cycle will focus more on certain procedures. For example, one cycle can be used basically for testing represented knowledge that was analyzed and gathered during a previous cycle. Any analysis carried out during this activity is based on the initial analysis done in activity 3.

It is again stressed that this activity focuses on the demonstration and analysis value of the prototype. Technical issues like machine requirements, screen design, etc. take a lower priority.

Resources.

During this activity ES, knowledge acquisition and knowledge engineering tools are used. Management, the KE and the experts are the main parties involved.

Activity A4 is complete when the KE has enough information to establish and achieve the following deliverables:

- A demonstration of the ES capabilities.
- An analysis of the expertise in the problem domain in order to determine the knowledge representation scheme.
- A representative problem from which Knowledge Engineering tool evaluations and selections can be done.

A5. DEMONSTRATION PROTOTYPE REVIEW

The purpose of this activity is to analyze the scope and impact that the ES will have on two areas: firstly business issues, eg. culture influences, and secondly technical issues, such as the Knowledge Engineering tool to be used, system interfaces, knowledge representation schemes and the like.

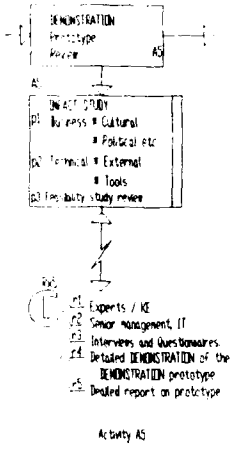
Procedures.

The KE conducts an impact study. which involves reviewing the scope and impact of the system in technical and business terms. In many instances it is found that the

initial problem definition and domain scope change and this differs from what at was first perceived. This change can have a different impact on the business and technical domain than was first envisaged.

Resources

The involvement of senior management during this activity is necessary because the impact that the proposed system can have may be felt throughout the whole organization as was the case of the development of XCON [14]. A report detailing the previous activities, leading to a recommendation to continue or abandon the project, is then compiled. Information like feasibility studies and duration estimates to management is crucial. With the revision of every demonstration prototype and production of every operational ES, information is produced to update the strategic plan [15,16] for the implementation of Knowledge based or Expert Systems.



Approval to continue the ES project from senior management will be based on the strategic plan, the needs of the business, economic considerations and the reports that the KE has produced.

7 CONCLUSION.

An overview of *MethoDex* to develop ESs has been described. This methodology is based on the conventional SDLC (the revised waterfall model) combined with a prototyping approach (the KE cycle)

The commercial use of ESs by the normal IT corporate function is growing rapidly. The days when AI specialists "hacked" away and produced efficient but "nearly impossible to maintain" AI systems are over. Functional and efficient approaches to developing ESs jointly and separately, in a clear and manageable way, are desperately needed. A great deal regarding the method of implementing this technology still needs to be done. This will be realised through experience. *MethoDex* tries to address this need, to form a basis on which the commercial systems development function of an organization can build its Expert Systems.

---

[1] Mellis, W. **Twaice: A Knowledge Engineering Tool**, *Information Systems*, Vol. 15, No. 1, Nov. (1990), pp. 137 - 150.

[2] Maletz, M.C. **Expert Systems: From the Research Laboratories to End-User Deployment**, *Expert Systems, Planning/Implementation/Integration*, Vol. 1, No. 1, Spring, (1989), pp. 38 - 43.

[3] Keyes J. **Why Expert Systems Fail**. *AIExpert*, Vol. 4, No. 11, Nov. (1989), pp.50 - 53.

[4] Kenny Jih W.J. **Comparing Knowledge Based and Transaction Processing Development**. *ASM/Journal of Systems Management*, May, (1990), pp. 23 - 28.

[5] Hays-Roth F., Waterman D.A., Lenat D.B. *Building Expert Systems*. Addison-Wesley Reading Massachusetts, (1983).

[6] Sprague K.S. **Cultivating a Prototyping approach to Expert Systems development.** *Expert Systems Planning/Integration/Implementation*, Vol. 2, No. 3, Fall (1990), pp. 37- 43.

[7] Apar P. **Towards structured Expert Systems development.** *Expert Systems with Applications*. Vol. 1, No. 1, (1990) pp. 63 - 70.

[8] Jarke N., Jeusfeld M., Rose T. **A Software Process Data Model for Knowledge Engineering in Information Systems.** *Information Systems*, Vol. 15, No. 1, (1990), pp. 85 - 116.

[9] Boehm B.W. *Software Engineering Economics*. Prentice-Hall, Inc., Englewood Cliffs, NY. (1981)

[10] Boehm B.W. **A Spiral Model of Software Development and Enhancement.** *Computer*, May, (1988), pp. 61 - 72.

[11] Liebowitz J. **When is a Prototype an Expert System ?** *Expert Systems*, Vol. 3, no. 1, Spring (1991) pp. 17 - 21.

[12] Maletz M.C. **Expert Systems: From the Research Laboratories to End-User Deployment.** *Expert Systems Planning/Implementation/Integration*, Vol. 1, no. 1, Spring (1989), pp. 38 - 43.

[13] Prerau D.S. *Developing and Managing Expert Systems: Proven Techniques for Business and Industry*. Addison Wesley, (1990), pp. 248 - 259.

[14] Sviokia J.J. **An Examination of the Impact of Expert Systems on the Firm: The Case of Xcon.** *MIS Quarterly*, June (1990), pp. 127 - 140.

[15] Moulin B. **Strategic Planning for Expert Systems.** *IEEE Expert*, Vol. 5, no. 2, April (1990), pp. 69 - 75.

[16] Meador C.L., Mahler E.G. **Choosing an Expert Systems Game Plan** *Datamation*, Aug. 1, (1990), pp. 64 - 69.

Towards Intelligent Databases

François Bry

ECRC, Arabellastraße 17, 81925 München 81, Germany
Francois.Bry@ecrc.de

Abstract. This article is a presentation of the objectives and techniques of deductive databases. The deductive approach to databases aims at extending with *intensional* definitions other database paradigms that describe applications *extensionally*. We first show how *constructive* specifications can be expressed with *deduction rules*, and how *normative* conditions can be defined using *integrity constraints*. We outline the principles of bottom-up and top-down query answering procedures and present the techniques used for integrity checking. We then argue that it is often desirable to manage with a database system not only database applications, but also specifications of system components. We present such *meta-level* specifications and discuss their advantages over conventional approaches.

1 Introduction

Deductive Databases have been studied since more than a decade. Theoretical issues have been investigated (see e.g. [28, 29, 30, 31, 65, 21, 8, 48, 64, 17, 18, 44, 45] for an overview), and experimental deductive database management systems have been and are still implemented (e.g. [54, 9, 23, 26, 32, 34, 51, 56, 66, 33, 68, 40, 52]). Industrial products are currently developed from research prototypes (e.g. [69]). This article is informal presentation of the notions and objectives of deductive databases. Instead of emphasizing technical aspects (that are explained in a number of articles and tutorials, e.g. [28, 29, 30, 31, 65, 21, 8, 17, 18]), we prefer to insist on the goals of the deductive approach to databases.

A first part of the presentation is devoted to recall how two complementary notions are used in deductive databases for *declaratively* specifying an application. On the one hand, *deduction rules* are used for *constructive* definitions. On the other hand, *normative* specifications are expressed through *integrity constraints*. We informally describe how deduction rules are evaluated for answering queries (see e.g. [17, 18, 1, 2, 4, 7, 55, 57, 60, 61, 67, 13]), and how integrity constraints are checked when the database is updated (see e.g. [17, 18, 15, 25, 39, 43, 47, 49, 53, 19]).

In a second part of the presentation, we argue that it is often desirable to manage with the database system, not only an application, but also specifications of components of the database sytem itself, the description of an application, or various kinds of interpretations of this application. We informally introduce a few such *meta-level* specifications, that rely on meta-programming [58, 62, 63, 59]. Finally, we briefly mention further applications of meta-level specifications towards enhanced database management systems.

2 An Introduction to Deductive Databases

A main trend in database research is the enhancement of data modeling facilities. Deductive database techniques aim at extending conventional, nondeductive databases, in which data are *extensionally* specified, with *intensional* definitions in form of *deduction rules* and *integrity constraints.*

Database management systems historically developed from file managers, in which applications are specified in terms of records and structured according to storage and retrieval criteria. Two data models were proposed at the end of the sixties/beginning of the seventies for improving the descriptions of applications: the hierarchical and the network data models. Like a file, a hierarchical or network database consists of records. However, in contrast to files, records are structured in trees and pointers express relationships between records. Both the hierarchical and the network data models have a major drawback: The pointers these data model rely upon make the design and the querying of databases rather difficult. Database users must be aware of rather complex networks even for posing simple queries.

The relational data model, defined by Codd [24] at the end of the seventies, overcomes this difficulty in an elegant manner: no pointers are used and the conceptual links between records (called tuples) are expressed through regular data. A relational database consists in a set of relations. Relations are set of tuples. The semantical relationship between tuples are expressed through the *values* they contain. Thus, for example, the presence of a same character string (say, a name) in a tuple of a "salary" relation and in a tuple of an "address" relation links salaries, addresses, and employee's names. Because they are *value-based,* relational databases can be interpreted in mathematics as logical theories consisting of formulas or, alternatively as logical models consisting of relations. Relational databases can be seen as more *declarative* than hierarchical or network databases since less knowledge of their internal structure is necessary for querying them. Indeed the knowledge of the relation's names, the so-called database schema, and, possibly, of some values occurring in tuples, suffices for posing queries.

2.1 Deduction Rules

Deductive databases can be seen as an extension of the relational model. In a relational database, the data are specified *extensionally.* That is, the tuples of a relational database are explicitly defined. Deductive databases in contrast, also give rise to specifying data *intensionally* by means of general properties, expressed using *deduction rules.* Consider for example the time-table of the Lufthansa airline. The Lufthansa direct flights from Munich to Paris can be specified by the following "flight" relation:

Monday	0725	0900	LH4356
Tuesday	0725	0900	LH4356
Wednesday	0725	0900	LH4356
Tursday	0725	0900	LH4356
Friday	0725	0900	LH4356
Saturday	0725	0900	LH4356

Monday	1110	1245	LH4384
Tuesday	1110	1245	LH4384
Wednesday	1110	1245	LH4384
Thursday	1110	1245	LH4384
Friday	1110	1245	LH4384
...	...	...	...

The first attribute (column) of this relation indicates the day of the flight, the second and third are the departure and arrival times, respectively, and the last attribute is the flight number. These eleven flights could be specified by the following two deduction rules that somehow "factorize" the data common to several tuples:

flight(D, 0725, 0900, lh4356) ← day(D), not D = sunday.
flight(D, 1110, 1245, lh4384) ← day(D), not D = saturday, not D = sunday.

As usual, character strings beginning with an upper case letter (e.g. D) are used for denoting (logical) variables. The membership of a tuple (called "fact" in deductive databases) "t" in a relation "r" is expressed by the term "r(t)". We assume that "day" denotes the relation containing the seven days of the week (monday, tuesday, etc.). Lower case letters are used for distinguishing these constant values from variables. The expression "day(D)" can be thus evaluated to the facts "day(monday)", "day(tuesday)", etc. The meaning of the first rule is that the facts "flight(monday, 0725, 0900, lh4356)", "flight(tuesday, 0725, 0900, lh4356)", ..., "flight(saturday, 0725, 0900, lh4356)" are derivable, i.e. are true facts in the database. In more technical terms,the variable D is (implicitly) universally quantified. The first deduction rule is thus a shorthand notation for the following formula:

$$\forall\, D \quad [\, (\mathrm{day}(D) \wedge D \neq \mathrm{sunday}) \;\Rightarrow\; \mathrm{flight}(D, 0725, 0900, \mathrm{lh4356})\,]$$

This simple example illustrates two important advantages of deductive databases compared with relational ones: (1) they require less storage, and (2) they give rise to more natural specifications. The possible size reduction is sometimes dramatic: An analysis of the time table of the Munich public transportation shows for example a reduction factor of about 200! Database applications whose data cannot be specified according to general principles do not benefit as much of deductive techniques. Most databases nevertheless contain some data that were implied from general laws (e.g. business rules, legislation, scientific laws, etc.) and therefore can benefit from deductive database techniques.

One could object that no deductive techniques are needed for achieving the factorization described above. This is true. There are indeed, for this example, two alternative ways to avoid the undesirable duplication of data using relational data structures. The first approach consists in splitting the original relation in two distinct relations, the first one giving the day and the flight number (which obviously is a key), the second relation giving the times and the flight numbers. A join then permits ones to reconstruct the original relation at query time. The second approach consists in using codes like in the following table for expressing on which days a flight is available.

Xe7	0725	0900	LH4356

Xe67	1110	1245	LH4384
...	...	...	...

In this relation, X stands for every day of the week, 6 for Saturdays, 7 for Sundays, Xe7 for every day except on Sundays, and Xe67 for every week days.[1]

We argue that both approaches have severe drawbacks. The first approach (the split of the original relation in two distinct smaller relation) examplifies an often criticized (although necessary) practice in relational database design: For reasons of storage (size) and coherency of the data (when updates are performed), the natural description of an application usually needs to be modified. The two rules given above as opposed achieve the same effect without compromising the natural character of the specifications. The second approach (the encoding of the days in the tuples) is very close to a specification by means of deduction rules. The difference however is that the encoding is a notation "unknown" to the database management system, while deduction rules are "understood" by a deductive database system for what they are. Such an encoding is specific to a given application and must be interpreted in the application programs, that is *outside* the database system. Deduction rules in contrast give rise to interpreting intensional knowledge *within* the database system.

Deduction rules can also be used in lieu of *relational views*. Views are in relational databases means for expressing predefined queries. One could for example define connecting flights using a view: A connecting flight form A to B is defined from a flight from A to C and a flight from C to B such that some conditions on the departure and arrival times in C, and on the location of the airport C are satisfied. A recursive definition give rise to specifying connections involving an indefinite number of flights. Such a definition is quite naturally expressed by the following deduction rule:

```
connection(D, T1, T2, [Nb]) ← flight(D, T1, T2, Nb).
connection(D, T1, T2, [Nb | L]) ← flight(D, T1, T3, Nb),
                                  connection(D, T3, T2, L),
                                  compatible(Nb, L).
```

The first rule specifies a connection consisting of one single flight. The list of flight involved in this connection ([Nb]) thus contains only one flight number. The second rule "links" a flight to a connection and extends its list of flight numbers. The predicate "compatible" is assumed to express whether times and airports are compatible in a connection. It might be specified intensionally by means of deduction rules, or extensionally by a relation. Recursive specification are important in practice for specifying several natural properties that apply on an indefinite number of object. Another example is the definition of a "bill of material": the price of a complex object is obtained by summing up the prices of its parts, whose prices are in turn similarly defined. Like for flight connections, it is desirable to have a specification at our disposal which is not limited to a given number of components (e.g. flights or parts). It has often been observed that recursive specifications are hardly avoidable in real life applications.

Deduction rules thus are very similar to relational views. Since the first relational database systems were not capable of handling recursive views, deduction rules are

[1] This representation is taken from the time table booklet published by Lufthansa.

often seen as the extension of relational views to recursion. In our opinion, deduction rules are more than extended views. Views are not handled like regular data, i.e. tuples and relations, in a relational database management systems, while deduction rules should be seen as first class citizen in a deductive database system. This means that all the facilities that are provided by the system for storing, retrieving, updating, and querying extensional specifications (i.e. facts) should also be applicable to intensionally defined data (i.e. data defined by deduction rules) and to the intensional specifications themselves. The full realization of this objective is still the subject of active research.

2.2 Remarks on the Language of Deduction Rules

The deduction rules specifying connecting flights (cf. previous section) contain complex, nested terms, namely lists. It is often believed that nested terms and term constructors should be prohibited in deductive databases. We think that nested terms are needed (as in the above example). Moreover, the known techniques are (almost) sufficient to acommodate them like flat, so-called first-normal form facts. It is probably the concept of *Datalog*, i.e. the language of rules with flat terms and no negation, which has widespread the idea that deductive databases should only specify first-normal form tuples.

In deductive databases, the same form of negation is needed as in relational databases. This negation has been formalized in various manner and under different names (negation as failure, non-monotonic negation, negation according to the closed-world assumption, etc.). Common to these formalizations is the basic notion that an expression can be considered as false if it cannot be proved. This interpretation of negation is a rather intuitive form of reasoning. This is this way of thinking that leads us to conclude, for example, that there are no direct flights from Munich to Trondheim if we do not find any in the time table. Although there is a general agreement on the semantics of this form of negation for relational databases, it is not always clear how to formalize it in deductive databases. Rules like the following ones are difficult to interpret, indeed:

$$a \leftarrow \text{not } b.$$
$$b \leftarrow \text{not } a.$$

"a" should be derivable only if "b" is not derivable, and "b" should be nonderivable only "a" is also nonderivable. Various more or less complex, more or less intuitive proposals have been made for giving convincing interpretations to such examples (and to more sophisticated ones) as well as for defining query answering procedures according to (some of) these interpretations. The problem is not yet completely solved and is still investigated. There is however a general agreement on the semantics of negation in so-called *stratified* deductive databases (or logic programs). The basic idea of stratification is to partition hierarchically the definitions of predicates, such that no predicate definitions refers to the negation of a predicate defined in a higher strata. Since one might have to deal with incompletely, or even incorrectly specified databases – for example for debugging at design time –, it is desirable to have a semantics (and the corresponding answering procedures) at our disposal which does not impose any syntactical restrictions such as stratification.

There is however a syntactical restriction which is desirable, that of *range restriction*. Range restriction basically requires that any variable occurring in a negated expression in a query or in the body (i.e. the right hand side) of a rule also occurs in a unnegated, positive expression. Thus, "p(X), not q(X)" is range-restricted, but "p(X), not q(X, Y)" is not because the variable Y has no (positive) range. Since, due to the interpretation of negation, negative expressions are absent from the database, range restriction is needed for ensuring that the variables occurring in a query or in a rule body can be assigned values from subexpressions occurring in this query or rule.

2.3 Integrity Constraints

Deduction rules give rise to generating new facts from a database, i.e. deduction rules express *constructive specifications*. In contrast to deduction rules, *integrity constraints* are used for expressing non-constructive, *normative* specifications. Such specifications are needed for ensuring that some properties remain satisfied when data are updated. The following integrity constraint for example states that no flights are allowed to land after 23:00:

$$\forall\ D\ T1\ T2\ Nb\quad [\ flight(D, T1, T2, Nb) \Rightarrow T2 \geq 2300\ ]$$

Any attempt to specify a flight landing after 23:00 would lead to a violation of this integrity constraint. This violation would be reported to the database user who could then either modify the update, or, if it appears to be no more valid, the integrity constraint instead. An integrity constraint can thus be viewed as a yes/no query which is evaluated when the database is updated. Integrity constraints are needed not only for specifying negative properties, as in the previous example, but also for stating disjunctive or existential conditions, like in the following examples stating that at least one of two flights must be recorded (i.e. specified) in the database, and that there exists at least one day on which there is a flight, respectively:

$$flight(saturday, 0700, 0745, lh0345) \vee flight(saturday, 0735, 0810, lh0346)$$
$$\exists\ D\quad [\ day(D) \wedge flight(D, 0700, 0745, lh0345)\ ]$$

Although marketed database management systems can only maintain very limited types of integrity constraints (if at all!), normative specifications are important in all kinds of database applications. Integrity constraints are expressed and maintained through application programs in current databases, that is *outside* the scope of the database system. This is undesirable because this makes the specification and the maintenance of integrity constraints a (generally complex) programming task. In deductive databases, this is part of the database design, for which tools should be available [16]. Integrity constraints are not declaratively specified but are expressed by means of imperative programs. Moreover these programs usually combine the *specifications* of the normative conditions and their efficient *evaluation*. In deductive databases in contrast, one only has to specify integrity constraints. Their efficient evaluation is left to the database management system (cf. Section 4 below). This is not only more convenient for the database designer. This also ensures that integrity constraints are efficiently checked. This is hardly the case when application programs

are modified for acommodating the modifications of integrity constraints that are unavoidable in any real life applications.

Range restriction is needed for integrity constraints like for deduction rules. A universal quantification $\forall$ X F[X] is range restricted if the expression F[X] is of the form R[X] $\Rightarrow$ G[X] and if X appears positively in R (cf. [10] for a precise definition). Thus $\forall$ X [p(X) $\Rightarrow$ q(X)] is range restricted, while $\forall$ X [($\neg$ p(X)) $\Rightarrow$ q(X)] is not. An existential constraint $\exists$ X F[X] is range restricted if F[X] is of the form R[X] $\wedge$ G[X] and if X appears positively in R[X] [10]. Range restriction ensures that only updates affecting expressions occurring in a constraint (directly or indirectly through deduction rules) might violate this constraint. This is an essential condition for an efficient integrity checking (cf. Section 4). It is worth noting that range restriction is a very natural requirement: in natural languages, it is almost impossible to express properties that are not range restricted. Moreover, formulas that are not range restricted have "semantically equivalent" counterparts that are range restricted.

2.4 Constraints as Rules

Deduction rules can be used for expressing integrity constraints in two different ways. The first one consists in expressing quantifiers by means of rules, the second approach, in rewriting the integrity constraint as special rules. The following deduction rule express a range-restricted universal quantification:

forall(X, R => F) $\leftarrow$ not (R, not F).

Consider for example the following universal formula: $\forall$ X p(X) $\Rightarrow$ q(X). It would be expressed as "forall(X, p(X) => q(X))" using the formalism defined by the above given rule. This expression evaluates to true if and only if it is impossible to satisfy the conjunctive query "p(X), not q(X)", i.e. to find a value X in the relation "p" which is not also in the relation "q". The deduction rule given above thus specifies a constructive evaluation of range restricted universally quantified expressions [12]. Existential quantifications are even easier to express in the formalism of deduction rule:

exists(X, F) $\leftarrow$ F.

Instead of relying on the above given rules for quantifiers, one can also directly rewrite the integrity constraints as rules. An integrity constraint C is expressed as a rule, called *denial*, corresponding to "false $\leftarrow$ not C". The examples of integrity constraints given above lead thus to the following denials:

false $\leftarrow$ flight(D, T1, T2, Nb), T2 > 2200.
false $\leftarrow$ not flight(saturday, 0700, 0745, lh0345),
not flight(saturday, 0735, 0810, lh0346).
false $\leftarrow$ not (day(D), flight(D, 0700, 0745, lh0345))

The two approaches are in fact the two sides of a same coin. The second representation is obtained from the first by partial evaluation (or partial deduction) [42, 35, 36, 37, 41] of the rules specifying quantifiers in the integrity constraints.

3 Query Answering

Queries are usually answered against the constructive specifications contained in the database, i.e. against the facts and deduction rules. Standard query answering methods do not make use of integrity constraints. Two complementary techniques can be applied in standard query answering: bottom-up (or forward) or top-down (or backward) reasoning. Bottom-up reasoning procedures basically consist in repeating the following as long as new facts are obtained: the bodies of all rules are evaluated against the explicitly stored facts, and the corresponding facts specified by the heads (i.e. the left hand side) of the rules are added to the database (in a special area). Consider for example the following database which can be interpreted as follows. "f(X, Y)" means that "X" is the father of "Y"; the odd (even, resp.) numbers are in a father-child relationship, and this relationship has circles on letters ("a" and "b" as well as "c" and "d" are "fathers" of each other); "g(X, Y)" means that "X" and "Y" belong to the same generation.

g(X, Y) ← f(FX, X), g(FX, FY), f(FY, Y).	f(1, 3)	f(2, 4)	f(a, b)
g(1, 2)	f(3, 5)	f(4, 6)	f(b, a)
g(a, c)		f(6, 8)	f(c, d)
			f(d, c)

The facts "g(1, 2)" and "g(a, c)" give rise to deriving "g(3, 4)", "g(b, d)", and "g(5, 6)" using the deduction rule. Bottom-up reasoning on this database leads to generating these facts in stages:

Stage 1:	g(3, 4)	g(b, d)
Stage 2:	g(3, 4)	g(b, d)
	g(5, 6)	g(a, c)
Stage 3:	g(3, 4)	g(b, d)
	g(5, 6)	g(a, c)
		g(b, d)

The next round derives the same facts are those proven at stage 3. For restricting the repeated derivation of already proven facts, one can require that at least one of the facts produced at the previous stage is used in a proof. This refined procedure is called in the database community, the *semi-naïve* method, while the straightforward, redundant method is called *naïve*. The naïve and semi-naïve methods terminate as soon as no new facts are derived. It is not possible to completely avoid a repeated generation of some facts, for a same fact can have several distinct proofs. Using bottom-up reasoning for answering a query basically consists in generating all derivable facts from the database, and then in evaluating the query against the resulting, extended set of facts. There are methods for restricting to some extent and in some cases this "blind" generation. However, it is an inherent feature of bottom-up reasoning not to make use of the posed query in trying to answer it: bottom-up reasoning is not "goal directed". It is worth emphasizing that the naïve and semi-naïve methods compute *sets* at each stages and that set-oriented techniques from relational system can be applied for computing these sets. An efficient processing of quantifiers, negation, and disjunctions that are frequent in integrity constraints and deduction rules requires to refine over the traditional techniques of relational algebra [10].

Top-down reasoning procedures overcome this drawback by reasoning backward from the posed query. Consider once again the father–generation database given above and the query "g(3, X)" asking for all Xs that are in the same generation as 3. The only solution is 4, since 1 and 2 are fathers of 3 and 4, respectively and belong themselves to a same generation. Reasoning backwards from the query "g(3, X)" consists in selecting a rule whose head unifies with the query. In our case, there is only one candidate rule. The unification of its head with the query binds the variables in its body resulting in the following conjunctive subquery: f(FX, 3), g(FX, FY), f(FY, Y). The first conjunct "f(FX, 3)" has one single solution which binds the variable FX to 1. The next conjunct (or subquery) to evaluate is "g(1, FY)". It can be answered either against the facts, or using once again the deduction rule in which case the same process is repeated.

Top-down reasoning can be formalized in terms of bottom-up reasoning by relying on the formalism of deduction rules as follows [13]:

fact(X) ← query(X), rule(X ← Y), answer(Y).
query(Y) ← query(X), rule(X ← Y).
query(Y1) ← query((Y1, Y2)).
query(Y2) ← query((Y1, Y2), answer(Y1).
answer(X) ← query(X), fact(X).
answer((Y1, Y2)) ← query((Y1, Y2)), answer(Y1), answer(Y1).

Assume that these rules are evaluated bottom-up and that the predicate "fact" ("rule", respectively) range over the facts (deduction rules, resp.) stored in the database. The first rule select a deduction rule the head of which unifies with a query, and, if an "answer" (a predicate defined by other rules) is found, generates a fact. In the formalism of deduction rules used here, unification does not have to be redefined: it is already provided by this formalism. The second rule generates a (generally conjunctive) query by unifying a query with the head of a rule. The third and fourth rules split conjunctive queries; the last two rules derive conjunctive answers by conjuncting already generated answers. The query "g(3, X)" is answered as follows by processing the deduction rules given above with the semi-naïve method:

Stage 1: query((f(ZX, 3), g(ZX, ZY), f(ZY, Y)))
Stage 2: query(f(ZX, 3))
...
Stage 6: answer(g(3, 1)) query((f(ZX, 1), g(ZX, ZY), f(ZY, Y)))
Stage 7: query((g(1, ZY), f(ZY, Y)))
...
Stage 11: answer(g(3, 4))

The above mentioned, rule-based specification of top-down reasoning is interesting for several reasons. Firstly, since it is expressed in terms of bottom-up reasoning, it is easily amenable to set-oriented computations. This is important for the sake of efficiency in databases. Secondly, the above specified top-down procedure is *complete,* more precisely *exhaustive:* If there are finitely many answers, it computes all of them and terminates; if there are infinitely many answers (in presence of function symbols), each single answer in computed in finite time. The top-down reasoning

procedure generally used in logic programming, SLD resolution, in contrast might loop in presence of recursive deduction rules. Termination (or exhaustivity) is important in databases, for database users as opposed to programmers cannot be made responsible of termination of the queries they pose to a database. Finally, the specification given above provides with a simple formalization of the Alexander or Magic Set rewriting methods [1, 2, 57, 7, 55, 4, 21, 8]: these rewritings are obtainable from the rule-based specification given above by partial evaluation (or partial deduction) [42, 35, 36, 37, 41]. These points are discussed in more detail in [13]. [61] also shows, from a different angle, that the Alexander and Magic rewritings in fact implement top-down reasoning by means of deduction rules that are evaluated bottom-up.

We would like to conclude this section on deductive database query answering methods with some remarks. Firstly, although "goal directedness" in general is important for efficiency, there are cases where the overhead resulting from generating and managing subgoals does not pay off. In theses cases, that still remain to be fully characterized, a bottom-up reasoning with the semi-naïve method is more efficient than a top-down procedure like the above specified one or Magic Set. Secondly, it is in some cases preferable to compute all derivable facts beforehand instead of generating the needed one for each query at query time. In these cases as well, bottom-up reasoning with the semi-naïve method is preferable. Finally, there has been proposals to use integrity constraints either for speeding up or for enhancing query answering (cf. e.g. [22, 46, 50, 14]). These approaches are very promising. They often give rise to more informative answers than conventional query answering methods.

4 Integrity Checking

Integrity constraints can be seen as yes/no queries (cf. Section 2.3). They can therefore be evaluated like regular queries. This is however often inefficient. Integrity constraints indeed are to be checked only after updates, and updates usually do not affect the whole of a database but only a limited part of it. For the sake of efficiency, it is desirable to check only those integrity constraints that might be affected by an update. Various integrity checking methods have been proposed that all rely on similar principles. Let us illustrates the techniques common to these so-called "integrity checking" methods on an example. Consider an integrity constraint requiring that all employees working for the sales department speak English:

$$\forall X [(empl(X) \wedge works\text{-}for(X, sales\text{-}dept)) \Rightarrow speaks(X, english)]$$

Any update to the facts and deduction rules that have no effect on the predicates occurring in this constraint cannot violate it. It is worth noting that this only holds if integrity constraints are range restricted. The insertion of any fact, say "p(a)" might violate a non range restricted constraint such as C: $\forall X\ q(X)$. If "a" did not occur in the database before the insertion of "p(a)", C indeed does not hold after the change. Whether an update might affect the definition of a relation can be specified using deduction rules as follows:

potential-update(H, Sign) ← rule(H ← B), potential-update(P, Sign)
potential-update((C1, C2), Sign) ← potential-update(C1, Sign).

potential-update((C1, C2), Sign) ← potential-update(C2, Sign).
potential-update(not F, Opp-Sign) ← potential-update(F, Sign),
opposite(Sign, Opp-Sign).
potential-update(F, +) ← insert(F).
potential-update(F, −) ← remove(F).

Let us comment this specification starting from the last two rules. The insertion (removal, resp.) of a fact F induces a "potential-update" on F with positive (negative, resp.) polarity. Negation changes the polarity of a potential update: For example, if "p(a)" is a potential removal, the negative information "not p(a)" is potentially inserted. The second and third rules specify that potential updates of conjuncts induce potential updates of conjunctions with same polarity. The first rule propagate potential insertions through deduction rules. Thus, if "p(a)" is inserted, the conjunction "(pa), q(a))" is a potential insertion. In presence of a rule "r(X) ← p(X), q(X)." "r(a)" is in turn a potential insertion.

All integrity checking method rely on analyses of possible (or actual) consequences of updates similar to the computation of potential updates which is specified above by means of deduction rules. This is quite intuitive when integrity constraints are expressed as denials. Integrity checking then indeed reduces to verifying whether "false" will become derivable after an update. Denials that cannot give rise to proving "false" can be filtered out by rules like the above mentioned ones, for "false" is derivable after an update only if "potential-update(false, +)" holds.

The analyses performed by the various integrity checking methods in some cases consider, in other cases ignore the values of the attributes. They sometimes perform bottom-up, sometimes top-down reasoning on the deduction rules, or on rules used for specifying the integrity constraints (cf. e.g. [43, 25, 39, 15, 17, 18] and [19] for an overview). Some methods, e.g. [53, 25, 15], simplify the integrity constraints with respect to updates. Such simplifications can be formalized as a partial evaluation (or partial deduction) [42, 35, 36, 37, 41] of deduction rules similar to those specified above.

In the rule-based specification of potential updates which is given above, we assume that the updates are specified as sets specified by the relations "insert" and "remove". It is worth noting that these relations can be defined intensionally by deduction rules as well as extensionally by means of facts. One could for example specify an update by the following rule:

insert(speaks(X, english)) ← nationality(X, british).

This rule is rather similar to the deduction rule "speaks(X, english) ← nationality(X, british)". The difference is that it forces the explicit storage of facts in the database, while the deduction rule for "speaks" does not. Integrity constraints can as well be defined on the predicates "insert" and "remove". The following integrity constraint for example forbids to fire of employees who work for the sales department:

∀ X [works-for(X, sales-dept) ⇒ ¬ remove(emp(X))]

5 Deduction Rules for Specifying System Components

In the previous sections, we have outlined how query answering and integrity checking procedures can be specified by means of deduction rules. The technique,which was used is known as *meta-programming*, for the variables in these deduction rules do not range, as in ordinary rules, over application data but instead over expressions (i.e. integrity constraints or rules) that describe the application data. We have pointed out that rewriting methods used for answering queries and evaluating integrity constraints can be seen as resulting from the partial evaluation (or partial deduction) of rule-based specifications. In this section, we first argue that it is beneficial to specify and implement some components of a database management system in this way. Then, we suggest further applications of this approach.

A first advantage of specifying components of a database management system using deduction rules and partial evaluation is the uniformity of the approach. Instead of implementing several rewriting methods for, say, recursive query processing (e.g. [1, 57, 2, 7, 4]), for simplifying integrity constraints (e.g. [43, 25, 15]), for query optimization (e.g. [46, 22, 10, 11]), etc. one could generate them automatically from the rule-based specifications using techniques as proposed in [58, 62, 42, 27, 35, 36, 37].

System components declaratively specified using deduction rules would probably be easier to prove correct and to maintain than conventional programs. Moreover, the very maintenance and updating tools provided by the database management system (e.g. integrity checking) could be applied to maintaining those system components that are specified in terms of deduction rules.

Specifying system components using deduction rules would in addition contribute to enhance the extensibility of the system. It is indeed easier to extend a set of deduction rules with additional rules for novel functionalities (e.g. additional query optimization strategies) than to extend a conventional program.

To which extent this approach is applicable in designing database management system is not yet known. The approach we suggest has however already been applied, more or less consciously, in many system prototypes that have been developed during the last years, e.g. [41, 68]. From discussions we had with designers of various database system prototypes (e.g. [54, 3, 23, 26, 33, 34, 40, 66, 68, 20, 69]) we gained the impression that meta-programming techniques are rather widely applied, although often quite unconsciously, in implementing database systems. The systematic investigation of this techniques for database system design is, we think, a promising direction of research.

Deduction rules can also be used for specifying data models and query languages. This is a widespread practice in logic programming to specify an interface model or language by means of rules. This can be done in deductive databases as well either for specifying a semantic data model (e.g. a entity-relationship model), or for specifying a query language (e.g. a SQL-like language). Rules can also be used for mapping complex objects on lower level data structures. Deductive databases are often criticized for being, like relational databases, value-based, and for not providing with object identities. Identities are "logical pointers" that give rise to naming objects [5, 6]. Extending the paradigm of logic programming and deductive facilities with identities is a promising issue. We think, this is the key issue to solve for bringing closer together both paradigms of deductive and object-oriented databases.

Deduction rules can finally also be used for interpreting the data stored in a database in various manners. Rules can be specified for various forms of reasoning that can be needed for some applications (e.g. hypothetical or probabilistic queries). Non-standard query answering methods (e.g. [62, 22, 46, 14]) often have been specified using meta-programming techniques.

6 Conclusion

This article has introduced and discussed the goals and techniques of deductive databases. We outlined how deductive databases give rise to *declaratively* specifying both, *constructive* and *normative* aspects of an application, using *deduction rules* and *integrity constraints*, respectively.

We informally presented bottom-up and top-down, set-oriented query answering methods, and we introduced to the principles upon which integrity checking methods are based. We have shown that deduction rules are not only useful for specifying database applications, but can also serve to specify and implement components of a database management system.

We finally argued that this approach is of interest for several reasons: It gives rise to a more uniform system design, system components implemented this way are easier to maintain; system extensibility is made easier.

Finally, we suggested further applications of this approach towards enhanced database systems.

References

1. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.: Magic Sets and Other Stange Ways to Implement Logic Programs. Proc. 5th ACM SIGMOD-SIGART Symp. on Principles of Database Systems (1986)
2. Bancilhon, F., Ramakrishnan, R.: An Amateur's Introduction to Recursive Query Processing. Proc. ACM SIGMOD Conf. on the Management of Data (1986)
3. Beierle, C.: Knowledge Based PPS Applications in PROTOS-L. Proc. 2nd Logic Programming Summer School (1992)
4. Beeri, C.: Recursive Query Processing. Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1989) (tutorial)
5. Beeri, C.: A Formal Approach to Object-Oriented Databases. Data & Knowledge Engineering 5 (1990) (Invited paper. A preliminary version of this article appeared in the proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases)
6. Beeri, C.: Some Thoughts on the Evolution of Object-oriented Database Concepts. Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (1993)
7. Beeri, C., Ramakrishnan, R.: On the Power of Magic. Proc. 6th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1987)
8. Bidoit, N.: Bases de Données Déductives. Armand Colin (1992) (in French)
9. Bocca, J.: On the Evaluation Strategy of Educe. Proc. ACM SIGMOD Conf. on the Management of Data (1986)
10. Bry, F: Towards an Efficient Evaluation of General Queries: Quantifier and Disjunction Processing Revisited. Proc. ACM SIGMOD Conf. on the Management of Data (1989)
11. Bry, F.: Logical Rewritings for Improving the Evaluation of Quantified Queries. Proc. Int. Conf. Mathematical Fundamentals of Data Base Systems (1989)

12. Bry, F.: Logic Programming as Constructivism: A Formalization and its Application to Databases. Proc. 8th ACM-SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1989)
13. Bry, F.: Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled. Data & Knowledge Engineering 5 (1990) (Invited paper. A preliminary version of this article appeared in the proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases)
14. Bry, F.: Constrained Query Answering. Proc. Workshop on Non-Standard Queries and Answers (1991)
15. Bry, F., Decker, H., Manthey, R.: A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases. Proc. 1st Int. Conf. on Extending Database Technology (1988)
16. Bry, F., Manthey, R.: Checking Consistency of Database Constraints: A Logical Basis. Proc. 12th Int. Conf. on Very Large Databases (1986)
17. Bry, F., Manthey, R.: Deductive Databases – Tutorial Notes. 6th Int. Conf. on Logic Programming (1989)
18. Bry, F., Manthey, R.: Deductive Databases – Tutorial Notes. 1st Int. Logic Programming Summer School (1992)
19. Bry, F., Manthey, R., Martens, B.: Integrity Verification in Knowledge Bases. Proc. 2nd Russian Conf. on Logic Programming (1991) (invited paper)
20. Cacace, F., Ceri, S., Crespi-Reghizzi, S., Tanca, L., Zicari, R.: Integrating Object-Oriented Data Modelling With a Rule-based Programming Paradigm. Proc. ACM SIGMOD Conf. on the Management of Data (1990)
21. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Surveys in Computer Science, Springer-Verlag (1990)
22. Chakravarthy, U.S., Grant, J., Minker, J.: Foundations of Semantic Query Optimization for Deductive Databases. In [48] (1988)
23. Chimenti, D., Gamboa, R., Krishnamurthy, R., Naqvi, S., Tsur, S., Zaniolo, C.: The LDL System Prototype. IEEE Trans. on Knowledge and Data Engineering 2(1) (1990) 76–90
24. Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. Comm. ACM 13 (1970) 377–387
25. Decker, H.: Integrity Enforcement on Deductive Databases. Proc. 1st Int. Conf. Expert Database Systems (1986)
26. Freitag, B., Schütz, H., Specht, G.: LOLA – A Logic Language for Deductive Databases and its Implementation. Proc. 2nd Int. Symp. on Database System for Advanced Applications (1991)
27. Gallagher, J.: Transforming Logic Program by Specializing Interpreters. Proc. European Conf. on Artif. Intelligence (1986) 109-122
28. Gallaire, H., Minker, J. (eds): Logic and Databases. Plenum Press (1978)
29. Gallaire, H., Minker, J., Nicolas, J.-M. (eds): Advances in Database Theory. Vol. 1. Plenum Press (1981)
30. Gallaire, H., Minker, J., Nicolas, J.-M. (eds): Advances in Database Theory. Vol. 2. Plenum Press (1984)
31. Gallaire, H., Minker, J., Nicolas, J.-M. (eds): Logic and Databases: A Deductive Approach. ACM Computing Surveys 16:2 (1984)
32. Haas, L. M., Chang, W., Lohman, G. M., McPherson, J., Wilms, P. F., Lpis, G., Lindsay, B., Pirahesh, H., Carey, M., Shekita, E.: Starburst Mid-Flight: As the Dust Clears. IEEE Trans. on Knowledge and Data Engineering (1990) 143–160

33. Jarke, M., Jeusfeld, M., Rose, T.: Software Process Modelling as a Strategy for KBMS Implementation. Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (1989)
34. Kiernan, G., de Maindreville, C., Simon, E.: Making Deductive Databases a Practical Technology: A Step Forward. Proc. ACM SIGMOD Conf. on the Management of Data (1990)
35. Komorowski, J.: Partial Evaluation – Tutorial Notes. North Amer. Conf. on Logic Programming (1989)
36. Komorowski, J.: Synthesis of Program in the Framework of Partial Deduction. Technical Report TR-81, Computer Science Depart. Åbo Akademi, Finland (1989)
37. Komorowski, J.: Towards Synthesis of Programs in the Framework of Partial Deduction. Proc. Workshop on Automating Software Design. XIth Int. Joint Conf. on Artif. Intelligence (1989)
38. Komorowski, J.: Towards a Programming Methodology Founded on Partial Deduction. Proc. 9th European Conf. on Artif. Intelligence (1990) 404–409
39. Kowalski, R. Sadri, F., Soper, P.: Integrity Checking in Deductive Databases. Proc. 13th Int. Conf. on Very Large Databases (1987)
40. Lefebvre, A., Vieille, L.: On Query Evaluation in the DedGin* System. Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (1989)
41. Lei, L., Moll, G.-H., Kouloumdjian, J.: A Deductive Database Architecture Based on Partial Evaluation. SIGMOD Records 19(3) (1990) 24–29
42. Lloyd, J., Shepherdson, J. C.: Partial Evaluation in Logic Programming. Jour. of Logic Programming 11 (1991) 217–242
43. Lloyd, J. W., Sonenberg, E. A., Topor, R. W.: Integrity Constraint Checking in Stratified Databases. Jour. of Logic Programming 1(3) (1984)
44. Lloyd, J. W., Topor, R. W.: A Basis for Deductive Database Systems. Jour. of Logic Programming 2(2) (1985)
45. Lloyd, J. W., Topor, R. W.: A Basis for Deductive Database Systems II. Jour. of Logic Programming 3(1) (1986)
46. Lobo, J., Minker, J.: A Metaprogramming Approach to Semantically Optimize Queries in Deductive Databases. Proc. 2nd Int. Conf. Expert Database Systems (1988)
47. Martens, B., Bruynooghe, M.: Integrity Constraint Checking in Deductive Databases Using a Rule/Goal Graph. Proc. 2nd Int. Conf. Expert Database Systems (1988)
48. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1988)
49. Moerkotte, Karl, S.: Efficient Consistency Control in Deductive Databases. Proc. 2nd Int. Conf. on Database Theory (1988)
50. Motro, A.: Using Integrity Constraints to Provide Intensional Responses to Database Queries. Proc. 15th Int. Conf. on Very Large Databases (1989)
51. Morris, K., Ullman, J. D., Van Gelder, A.: Design Overview of the NAIL! System. Proc. 3rd Int. Conf. on Logic Programming (1986)
52. Naqvi, S., Tsur, S.: A Logical Language for Data and Knowledge Bases. Computer Science Press (1989)
53. Nicolas, J.-M.: Logic for Improving Integrity Checking in Relational Databases. Acta Informatica 18(3) (1982)
54. Nicolas, J.-M., Yazdanian, K.: Implantation d'un Système Déductif sur une Base de Données Relationnelle. Research Report, ONERA-CERT, Toulouse, France (1982) (in French)
55. Ramakrishnan, R.: Magic Templates: A Spellbinding Approach to Logic Programming. Proc. 5th Int. Conf. and Symp. on Logic Programming (1988)

56. Ramakrishnan, R., Srivastava, D., Sudarshan, S.: CORAL: Control, Relation and Logic. Proc. Int. Conf. on Very Large Databases (1992)
57. Rohmer, J., Lescœur, R., Kerisit, J.-M.: The Alexander Method. A Technique for the Processing of Recursive Axioms in Deductive Databases. New Generation Computing 4(3) (1986)
58. Safra, S., Shapiro, E.: Meta-interpreters for Real. Information Processing 86. North-Holland (1986) 271–278
59. Sakama, C., Itoh, H.: Partial Evaluation of Queries in Deductive Databases. New Generation Computing 6 (1988) 249–258
60. Schmidt, H., Kiessling, W., Günther, H., Bayer, R.: Compiling Exploratory and Goal-Directed Deduction Into Sloopy Delta-Iteration. Proc. Symp. on Logic Programming (1987)
61. Seki, H.: On the Power of Alexander Templates. Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1989)
62. Sterling, L. S., Beer, R. D.: Meta-interpreters for Expert System Construction. Technical Report TR 86-122, Center for Automation and Intelligent System Research, Case Western Reserve Univ. (1986)
63. Takuchi, A., Furukawa, K.: Partial Evaluation of Prolog Programs and its Application to Meta Programming. Information Processing 86. North-Holland (1986) 415–420
64. Tsur, S.: A (Gentle) Introduction to Deductive Databases. Proc. 2nd Int. Logic Programming Summer School (1992)
65. Ullman, J. D.: Principles of Database and Knowledge-Base Systems. Vol. 1 and 2. Computer Science Press. (1988, 1989)
66. Vaghani, J., Ramamohanarao, K., Kemp, D., Somogyi, Z., Stuckey, P.: The Aditi Deductive Database System. Proc. NACLP Workshop on Deductive Database Systems (1990)
67. Vieille, L.: Recursive Query Processing: The Power of Logic. Theoretical Computer Science 69(1) (1989)
68. Vieille, L., Bayer, P., Küchenhoff, Lefebvre, A.: EKS-V1: A Short Overview. Proc. AAAI-90 Workshop on Knowledge Base Management Systems (1990)
69. Vieille, L.: A Deductive and Object-Oriented Database System: Why and How? Proc. ACM SIGMOD Conf. on the Management of Data (1993)

Combining Classification and Nonmonotonic Inheritance Reasoning: A First Step*

Lin Padgham and Bernhard Nebel

[1] Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
[2] German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-6600 Saarbrücken 11, Germany
linpa@ida.liu.se nebel@dfki.uni-sb.de

Abstract. The formal analysis of semantic networks and frame systems led to the development of *nonmonotonic inheritance networks* and *terminological logics*. While nonmonotonic inheritance networks formalize the notion of default inheritance of typical properties, terminological logics formalize the notion of defining concepts and reasoning about definitions. Although it seems to be desirable to (re-)unify the two approaches, such an attempt has not been made until now. In this paper, we will make a first step into this direction by specifying a nonmonotonic extension of a simple terminological logic.

1 Introduction

The formal analysis of early semantic network and frame formalisms led to the development of two different families of knowledge representation formalisms, namely, *nonmonotonic inheritance networks* [5] and *terminological logics* [12]. Nonmonotonic inheritance networks formalize the idea of default inheritance of typical properties. Terminological logics aim at formalizing the idea of defining concepts and reasoning with such definitions, for instance, determining *subsumption* relationships between concepts and *instance* relationships between objects and concepts—two kinds of inferences we will collectively refer to as *classification*.

Although these two forms of representation and reasoning may seem to be incompatible [2], it would of course be desirable to combine them. From the point of view of nonmonotonic inheritance networks, it would be interesting to have a richer description language for specifying classes and properties and to add the ability of *classifying* objects as belonging to some class. From the point of view

* This work has been supported by the the Swedish National Board for Technical Development (STU) under grant # 9001669, by the Swedish Research Council for Engineering Sciences under grant # 900020, by the German Ministry for Research and Technology (BMFT) under research contract ITW 8901 8, and by the European Community as part of the ESPRIT Working Group DRUMS-II.

of terminological logics, it is desirable to add forms of reasoning that deal with uncertain information. In fact, Doyle and Patil [3] argue that a representation system without such a facility is useless.

There are proposals to integrate some form of default inheritance in terminological logics since 1981 (see [14, 12]) and some terminological representation systems support forms of nonmonotonic inheritance, which appear to combine the two modes of reasoning in a "naive" way (e.g. [12]), however, leading to problems similar to the infamous "shortest path inference," as we will see in Section 3.

An attempt to combine classificatory reasoning and nonmonotonic inheritance that avoids the latter problem has been made by Horty and Thomason [4]. Although this approach comes closest to our intention of combining nonmonotonic inheritance and classification, there are some problems, for instance, the "zombie path" problem [7], the lack of an algorithm, and the computational intractability of the approach.

More recent approaches combine classificatory and nonmonotonic reasoning by integrating default logic into terminological logics [1]—without using specificity for conflict resolution, though—or they employ a form of preference semantics [13].

We will base our combination of inheritance reasoning and classification on the nonmonotonic inheritance reasoning approach by Padgham [10], which avoids the above mentioned shortcomings. In the following sections we introduce a restricted terminological logic extended by defaults, and discuss how the inheritance theory in [10] can be extended to include classification.

2 A Common Representational Base

In order to describe our approach, we first introduce a representation formalism that can be conceived as a restricted *terminological logic*.

We start with a set $\mathbf{A}$ of *atomic concepts* (denoted by $A, A', \ldots$) and a set $\mathbf{F}$ of *features* (denoted by $F, F', \ldots$) that are intended to denote single-valued roles. Additionally, we assume a set $\mathbf{V}$ of *values* (denoted by v, v') that are intended to denote atomic values from some domain. Based on this, *complex concept expressions* (denoted by C, C') can be built:

$$C \rightarrow \top \mid \bot \mid A \mid C \sqcap C' \mid F{:}v.$$

In order to *define* new concepts completely or partially, *terminological axioms* (denoted by θ) are used. *Assertions* (denoted by α) are employed to specify properties of objects ($x, y, z, \ldots \in \mathbf{O}$):

$$\theta \rightarrow A \sqsubseteq C \mid A \doteq C, \; \alpha \rightarrow x{:}C \mid F(x) \doteq v.$$

Knowledge bases are sets of such terminological axioms and assertions.

The semantics of this language is given in the usual set-theoretic way. An interpretation $\mathcal{I}$ is a tuple $\langle \mathcal{D}, \mathcal{V}, \cdot^{\mathcal{I}} \rangle$, where $\mathcal{D}$ and $\mathcal{V}$ are arbitrary non-empty sets that are disjoint, and $\cdot^{\mathcal{I}}$ is a function such that

$$\cdot^{\mathcal{I}}(\mathbf{A} \rightarrow 2^{\mathcal{D}}) \cup (\mathbf{F} \rightarrow 2^{(\mathcal{D} \times \mathcal{V})}) \cup (\mathbf{V} \rightarrow \mathcal{V}) \cup (\mathbf{O} \rightarrow \mathcal{D})$$

where we assume that the relation denoted by a feature is a *partial function* and that values and object identifiers satisfy the *unique name assumption*. Interpretations are extended to complex concept expressions in the usual way, e.g., $(C \sqcap C')^{\mathcal{I}} = C^{\mathcal{I}} \cap C'^{\mathcal{I}}$ and $(F{:}\, v)^{\mathcal{I}} = \{d \in \mathcal{D} | \, (d, v^{\mathcal{I}}) \in F^{\mathcal{I}}\}$.

An interpretation is called a *model* of a knowledge base, if all terminological axioms and assertions are *satisfied* by the interpretation in the obvious way, e.g., $A^{\mathcal{I}} = C^{\mathcal{I}}$ for $\theta = (A \doteq C)$. The specialization relationship between concepts (also called *subsumption*) and the *instance relationship* between object identifiers and concepts are defined in the obvious way. A concept C is subsumed by C', written $C \preceq C'$ iff $C^{\mathcal{I}} \subseteq C'^{\mathcal{I}}$ for all models $\mathcal{I}$ of the knowledge base. An object x is an instance of a concept C, written $x{:}\, C$, iff for all models $\mathcal{I}$ it holds that $x^{\mathcal{I}} \in C^{\mathcal{I}}$.

In order to express that an instance of a concept C typically has some additional properties, the syntax of terminological axioms is extended as follows:

$$\theta \rightarrow A \sqsubseteq C/D_1, \ldots, D_n \mid A \doteq C/D_1, \ldots, D_n,$$

where the D_i's are again concept expressions. These "default properties" do not influence the set-theoretic interpretation of concepts, but are intended to denote that an instance of A typically has the additional properties D_i. In terms of Padgham's [10] type model, *given an axiom* $A \sqsubseteq C/D_1, \ldots, D_n$, C represents the *core* of a type, while $C \sqcap D_1 \sqcap \ldots \sqcap D_n$ is the *default* of a type.

Using our simple representation formalism, we could classify concepts in the TBox, compute instance relationships between objects and concepts, and separately apply default inheritance in order to derive typical information about objects. In fact, this loose combination of classification and default inheritance was used profitably in a medical diagnosis application [15].

The network language we will use contains strict inheritance links "$\Rightarrow$", strict negative links "$\not\Rightarrow$", and default inheritance links "$\rightarrow$". In addition to the usual kind of nodes, depicted by a letter, we also allow for *defined* nodes, depicted by an encircled letter. The latter nodes are assumed to be defined by the conjunction of all nodes that are reachable by a single strict positive link. As an example let us consider the following small knowledge base (inspired by [2]):

$$\begin{aligned}
\texttt{Elephant} &\sqsubseteq \texttt{Mammal/legs}{:}\, 4, \texttt{color}{:}\, \texttt{grey} \\
\texttt{Hepatitis-Elephant} &\doteq \texttt{Elephant} \sqcap \texttt{infected-by}{:}\, \texttt{Hepatitis/color}{:}\, \texttt{yellow} \\
\texttt{Yellow-Elephant} &\doteq \texttt{Elephant} \sqcap \texttt{color}{:}\, \texttt{yellow} \\
x &{:}\, \texttt{Elephant} \\
\texttt{infected-by}(x) &\doteq \texttt{Hepatitis}
\end{aligned}$$

Using the abbreviations M, E, H, and Y for **Mammal**, **Elephant**, **Hepatitis-Elephant**, and **Yellow-Elephant**, respectively, and g, h, l, y for **color**: **grey**, **infected-by**: **Hepatitis**, **legs**: 4, and **color**: **yellow**, respectively, the network diagram corresponding to our small knowledge base would look like as in Figure 1.

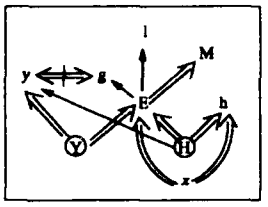


Figure 1: Shortest Path Problems

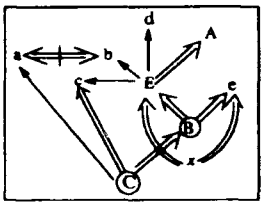


Figure 2: Example Network

3 Some Problems Combining Defaults with Classification

Using concept specificity for resolving conflicts among contradicting typical properties seems to be natural and desirable. Indeed, most proposals or already implemented systems seem to prefer this kind of conflict resolution. MacGregor, for instance, integrated a facility for "specificity-based defaults" [6, p. 393] into LOOM. The proposal by Pfahringer [12] is also an effort in this direction, employing a form of skeptical inheritance as defined in the area of nonmonotonic inheritance reasoning. From the limited descriptions of such approaches in the literature they appear to combine the two modes of reasoning in what we will call a "naive" way, which can be described as follows. Given an object x and a description D_0 of x, we first determine the set of most specialized concepts S such that x is an instance of all of the concepts in S. Based on this we determine additional typical properties of x using some inheritance strategy, which gives us a new (more specialized) description D_1, and we start the cycle again. We stop when a fixpoint is reached, i.e., D_i is equivalent to D_{i-1}.

The main problem with the "naive" approach is that it leads to results resembling the infamous shortest path inference. This can be illustrated by considering figure 2. If we begin by classifying x we get B. Default reasoning then gives $b, c, d.\neg a$ and a further round of classification gives C. we would now want by default to believe a, but this is blocked because we believe b and $\neg a$. However we observe that the default belief in a comes from a more specific type (C), than the default belief in b (which comes from E). We would therefore prefer to believe a than b. However we have previously commited to b because we reached it first.

4 A Default Inheritance Reasoning Framework

In this section we develop a formal framework for default inheritance reasoning. We will then generalize this in the following section so that it becomes a framework for combined classification and default reasoning. The framework that we develop is based on that presented in [10, 11]. The theory is very close to the skeptical inheritance theory of Horty *et al* [5] in terms of the conclusions reached[3], but instead of working with constructible, preempted and conflicted

[3] It does not however have the "zombie path" behavior criticized by Makinson and Schlechta [7].

paths, we work with notions of *default assumptions*, *conflicting assumptions* and *modification of assumptions* in order to resolve conflicts.

Given some initial information and an inheritance net, we first assemble all the default assumptions that may be possible, given this start point. We then find all the pairwise conflicting assumptions, and resolve the conflicts--starting with most specific nodes—by modifying one or both of the assumptions. Finally we add all our modified (and now consistent) assumptions together to obtain our set of conclusions.

Our formalization is based on an inheritance network, Γ, which is derivable directly from terminological axioms and assertions as defined in Section 2, and *labellings* which are mappings from the nodes in the network (the set $\mathbf{N}_\Gamma$) to values in the set $\{0, 1, -1, k\}$. The intuitive interpretation of such a labelling L is an information state concerning a hypothetical object where $L(X) = 1$ means that the object is an instance of the concept X, $L(X) = -1$ means that the object is not an instance, $L(X) = 0$ means there is no information concerning the instance relationship and $L(X) = k$ means there is contradictory information. The set $\{0, 1, -1, k\}$ forms a lattice w.r.t. information content such that $0 \leq 1 \leq k$, and $0 \leq -1 \leq k$. Similarly, the set of all labellings forms a lattice based on this ordering. In particular the *join* of two labellings, written $L_1 \sqcup L_2$, corresponds to the combination of the information content. The special labelling $\underline{0}$ is the labelling with all labels 0.

A labelling L is said to be *consistent* if it does not contain any node with a value of k. A pair of labellings is said to be *compatible* if their join is consistent, and *weakly compatible* if their join does not introduce any new inconsistency not present in one of the individual labellings.

Definition 1 *There is a* **strict positive path** *from X to Y in Γ, written $X \Rightarrow \sigma \Rightarrow Y$, iff exists W,Z: $(X{=}W \vee [X \Rightarrow \sigma \Rightarrow W] \in \Gamma) \wedge [Z \not\Rightarrow W] \in \Gamma \wedge (Y{=}Z \vee [Y \Rightarrow \sigma \Rightarrow W] \in \Gamma)$*

Definition 2 *There is a* **strict negative path** *from X to Y in Γ, written $X \not\Rightarrow \sigma \Leftarrow Y$, iff $[X \not\Rightarrow Y] \in \Gamma \vee (\exists W\text{: } [X \not\Rightarrow W] \in \Gamma \wedge [Y \Rightarrow \sigma \Rightarrow W] \in \Gamma)$.*

We define two particular kinds of labellings—*core labellings* (written X_c) and *default labellings* (written X_d) for a node X. A core labelling represents the necessary information for a node, while a default labelling for a node X represents the information typically associated with X.

Definition 3 *A core labelling for X (w.r.t. Γ), written X_c, is the minimal labelling which fulfills the following:*
$X_c(X) \geq 1$; and for all Y
$([X \Rightarrow \sigma \Rightarrow Y] \in \Gamma) \rightarrow (X_c(Y) \geq 1) \wedge ([X \not\Rightarrow \sigma \Leftarrow Y] \in \Gamma) \rightarrow (X_c(Y) \geq -1)$

Definition 4 *A* **default labelling** *for X (w.r.t. Γ), written X_d, is the minimal*[4] *labelling which fulfills the following:*
$X_d \geq X_c$; and for all Y $([X \rightarrow Y] \in \Gamma) \rightarrow (X_d \geq Y_c)$

[4] The ordering over labellings is the obvious one, given the ordering over node values.

Referring back to Figure 1, the core labelling for **Hepatitis-Elephant** would have values of $\{H = 1,\ h = 1,\ E = 1,\ M = 1\ all\ else = 0\}$. Its default labelling would contain $\{H = 1,\ h = 1,\ E = 1,\ M = 1,\ y = 1,\ g = -1,\ all\ else = 0\}$.

Referring again to Figure 1, we may wish to block that part of E_d (default elephant assumption) which concludes g (grey), but allow a modified assumption which concludes l (four legs). On the basis of default labellings we introduce the notion of *modified assumption* (written $X_{d'}$).

We define a *correct* modified assumption which intuitively allows removal only of entire branches from the full default assumption. Correctness ensures both consistency w.r.t. the network and also that (potentially) dependent properties are treated together.

Figure 3 gives some examples of correct and incorrect modified assumptions.[5]

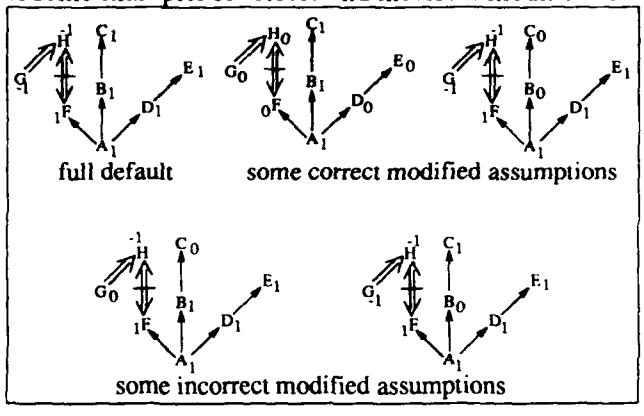


Figure 3: Correct and Incorrect Assumption Modifications

Definition 5 *A modified assumption, $X_{d'}$, is* **correct** *iff the following conditions hold for all Y in Γ:*

1. $([X \rightarrow Y] \in \Gamma) \rightarrow ((X_{d'}(Y) = 1) \rightarrow X_{d'} \geq Y_c$
2. $X_{d'}(Y) = 1 \rightarrow (\exists Z: [X \rightarrow Z] \in \Gamma \wedge Z_c(Y) = 1)\ X_c(Y) = 1$
3. $X_{d'}(Y) = -1 \rightarrow (\exists Z: [X \rightarrow Z] \in \Gamma \wedge X_{d'}(Z) = 1 \wedge Z_c(Y) = -1) \vee X_c(Y) = -1$.
4. $X_{d'} \geq X_c$.

When an assumption is modified it is always modified with respect to some other information with which it is in conflict. We thus introduce the notion of a modified assumption as a pair of labellings consisting of the default assumption labelling for the node and a *preference labelling* for the node (written P). The preference labelling captures all of the information which is to be preferred over the default assumption at that type. While it can in principal be an arbitrary labelling the preference labelling will for all interesting theories depend on both

[5] The concept of correctness is further motivated and explained in [10, p. 188–189], where it is dealt with as two separate concepts - groundedness and consistency.

the type network and the initially given information. There is no constraint on the preference labelling to be consistent.

The value of the preference labelling for a node determines the modified assumption for that node. If the preference labelling for a node X is not weakly compatible with X_c (indicating preferred disbelief in the concept), then the modified assumption will be empty. Otherwise the modified assumption is a labelling between the core and the default, w.r t. information content.

Definition 6 *A* **modified assumption** $X_{d'} = (X_d, P)$ *is* $\underline{0}$ *iff* X_c *is not weakly-compatible with* P*; otherwise* $X_{d'}$ *is the maximal labelling that is weakly-compatible with* P*, is a correct modification of* X_d *and* $X_c \leq X_{d'} \leq X_d$.

By joining a set of modified assumption labellings for a given network we can obtain a conclusion labelling for that network. We want to ensure that the preference labellings modify the default assumptions sufficiently to remove all conflicts so that we can obtain a consistent conclusion labelling. Preference labellings will be determined by the structure of the taxonomy together with the initial information, using principles such as specificity.

Each node in the network has its own preference labelling. We call this collection of preference labellings a *preference map*, written Θ. For a given network Γ, and a given initial labelling ψ, the preference map provides a preference labelling Θ_X for each node X in $\mathbf{N}_\Gamma$. Different inheritance theories can be compared with respect to the characteristics of their preference map. We will first characterize what we call a *well-formed* preference map, which can then be used as a base for defining a preference map for different kinds of theories, e.g. skeptical and credulous preference maps.

The characteristics that we capture in the definition of a well-formed preference map are that initial information is always preferred over default assumptions, more specific information is always preferred over less specific, unless the more specific information is unsupported, and that only supported (or reachable) modified assumptions are non-empty.

Definition 7 *A preference map* Θ *is* **well-formed** *for a network* Γ *and an initial labelling* ψ*, iff the following conditions are satisfied:*

1. $\Theta_X(X) < k$, *for all* $X \in \mathbf{N}_\Gamma$,
2. $\Theta_X \geq \psi$, *for all* $X \in \mathbf{N}_\Gamma$,
3. *there exists a strict partial ordering* $\ll$ *such that for all* $X \in \mathbf{N}_\Gamma$*: if* $\Theta_X(X) \leq 1$, *then* $\psi(X) = 1$ *or there is a* $Y \in \mathbf{N}_\Gamma$ *s.t.* $Y \ll X$ *and* $Y_{d'}(X) = 1$,
4. *if* X *is more specific*[6] *than* Y, *then* $X_{d'} \leq \Theta_Y$.

Θ^0 denotes the minimal well-formed preference map.[7]

To characterize a skeptical preference map we require in addition to well-formedness that each pair of modified assumptions are either compatible under

[6] For an exact definition of the notion of specificity used see [10, p. 142].

[7] Proof of the existence of a unique minimal well-formed preference map is due to Ralph Rönnquist, and can be found in [10] (where it is called a revision function).

well-formed preference, or that the preference labelling for each includes the other (forcing modification of each w.r.t. the other).

Definition 8 *A preference map Θ is* **skeptical** *for a network Γ and an initial labelling ψ, iff it is well-formed and for all $X, Y \in \mathbf{N}_\Gamma$: $(X_d, \Theta^0_X) \sqcup (Y_d, \Theta^0_Y)$ is consistent or $(\Theta_X \geq Y_d$ and $\Theta_Y \geq X_d)$.*

5 Integrating Classification into the Framework

As we saw in section 3 we should not simply interleave correct classification and correct default reasoning as this will lead to a certain arbitrariness in the results, and will give problems analogous to "shortest path" problems found in early approaches to default inheritance reasoning. We therefore take the approach of defining a single theory which includes both classificatory reasoning and default reasoning.

Condition 1 of Def. 7 for a well-formed preference map simply states that a preference labelling should not be inconsistent regarding the preference of the node for which it is a preference labelling. This is not affected by classification.

Condition 2 of Def. 7 captures that initial information should be preferred over all default assumptions. This is also a criteria which is clearly applicable to combined classificatory/default reasoning.

Condition 4 of Def. 7 says that we prefer assumptions associated with more specific, rather than less specific assumptions and also seems appropriate to retain unchanged.

The final condition of well-formedness, (condition 3) has to do with ensuring that a default assumption is empty unless we independently from it (and its results) believe in the base concept. Looking at Figure 4, and starting with information F, we clearly would not want to make any default assumptions regarding, for example, C or X.

In the default inheritance reasoning the support in the ordering of condition 3 of well-formedness, is shown by a labelling of 1 on a node in some "earlier" modified assumption. However if we include classification as a valid means of reaching a conclusion then support may come from not only single assumption(s) but from a set of assumptions, which, taken together provide the "evidence" for believing that type. In order to capture this formally, we define the notion of *support*.

Definition 9 *A set of labellings Π* **supports** *a node X iff for some labelling $L \in \Pi$: $L(X) = 1$ or for all $Z \in \mathbf{N}_\Gamma$: $((X_c(Z) = 1$ implies Π supports $Z)$ and $(X_c(Z) = -1$ implies that for all $L \in \Pi$: $L(Z) \leq -1))$.*

Note that the support required for default reasoning is simply a special case of this definition of support. We can now rewrite the third condition of well-formedness as follows:

3. *there is a strict partial ordering $\ll$ on the nodes in $\mathbf{N}_\Gamma$ such that $\forall X \in \mathbf{N}_\Gamma$: if $\Theta_X(X) \leq 1$, then $\{\psi\}$ supports X or there exists Π s.t. Π supports X and for all $L \in \Pi$ there exists a node $Y \in \mathbf{N}_\Gamma$ s.t. $L = Y_{d'}$ and $Y \ll X$.*

The additional criterion for a skeptical preference map ensures that any ambiguous conflicts which remain following application of specificity for conflict resolution will result in bilateral modification of the conflicting assumptions. This appears to be equally applicable to combined classification/default reasoning as it is to pure default inheritance reasoning. To illustrate the principle captured here we observe Figure 5. In the left-hand network, the default assumptions at E and B are both modified to avoid concluding $G, \neg F$ or $F, \neg G$ respectively.

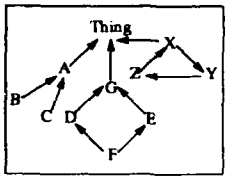


Figure 4: Support

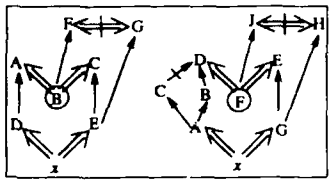


Figure 5: Ambiguous Conflicts

In the right hand network, the default assumptions at C and B are modified leading to no conclusion regarding D, and consequently no classification of F. However, because $\{B, C\}$ is an ambiguous conflict, the classification F would be made in some credulous extension, and we therefore allow it to cause modification to the assumption at E regarding H. Thus the conclusion for this figure will be A, B, C, G, E. We note that this is different than the extension given by Horty's method which will also include H and $\neg J$. H and $\neg J$ are not in the intersection of credulous extensions, and thus we would argue that they should not be in the skeptical extension. This difference is a result of the different treatment of ambiguous conflict in the two methods, where our approach is what has been referred to as "ambiguity propagating" in order to avoid the oddities of "zombie paths" [7].

6 Discussion

We have shown how a minor change to a theory for default inheritance gives a theory of combined classification and default inheritance for a very restricted terminological logic. Whilst this language may be too restricted for real applications it allows us to obtain a clearer understanding of the complex interaction between default inheritance and classification. This provides a firm basis on which we can begin to experiment with the addition of some greater expressivity. It may well be necessary to limit the expressivity of terminological languages with defaults, not so much because of tractability problems as most terminological languages are already intractable [8], but because of problems with "conceptual complexity". However there are certainly some applications which require defaults but whose other requirements on expressivity are limited [15]. The approach described in this paper provides a start for investigating languages and associated reasoning mechanisms for such applications.

The theory developed here is based on a skeptical inheritance theory with a polynomial algorithm. Considering the minor nature of the change required to

incorporate classification into this theory, the algorithm should also be directly modifiable to include classification.

References

1. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In Nebel et al. [9], pages 306–317.
2. R. J. Brachman. 'I lied about the trees' or, defaults and definitions in knowledge representation. *The AI Magazine*, 6(3):80–93, 1985.
3. J. Doyle and R. S. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–298, Apr. 1991.
4. J. F. Horty and R. H. Thomason. Boolean extsionsins to inheritance networks. In *Proceedings of the 8th National Conference of the American Association for Artificial Intelligence*, pages 633–639, Boston, MA, Aug. 1990. MIT Press.
5. J. F. Horty, R. H. Thomason, and D. S. Touretzky. A skeptical theory of inheritance in nonmonotonic semantic networks. In *Proceedings of the 6th National Conference of the American Association for Artificial Intelligence*, pages 358–363, Seattle, WA, July 1987.
6. R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, San Mateo, CA, 1991
7. D. Makinson and K. Schlechta. Floating conclusions and zombie paths: Two deep difficulties in the "directly skeptical" approach to defeasible inheritance nets. *Artificial Intelligence*, 48(2):199–211, Mar. 1991.
8. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
9. B. Nebel, W. Swartout, and C. Rich, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference*, Cambridge, MA, Oct. 1992. Morgan Kaufmann.
10. L. Padgham. *Non-Monotonic Inheritance for an Object-Oriented Knowledge Base.* PhD thesis, University of Linköping, Linköping, Sweden, 1989. Linköping Studies in Science and Technology. Dissertations No. 213.
11. L. Padgham. Defeasible inheritance: A lattice based approach. *Computers and Mathematics with Applications*, 23(6-9):527–541, 1992. Special Issue on Semantic Nets.
12. P. F. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. MacGregor, W. S. Mark, D. McGuinness, B. Nebel, A. Schmiedel, and J. Yen. Term subsumption languages in knowledge representation. *The AI Magazine*, 11(2):16–23, 1990.
13. J. Quantz and V. Royer. A preference semantics for defaults in terminological logics. In Nebel et al. [9], pages 294–305.
14. J. G. Schmolze and R. J. Brachman, editors. *Proceedings of the 1981 KL-ONE Workshop*, Cambridge, MA, 1982. Bolt, Beranek, and Newman Inc. BBN Report No. 4842.
15. T. Zhang and L. Padgham. A diagnosis system using inheritance in an inheritance net. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent systems*, Knoxville, TN, Oct. 1990. North-Holland.

Mechanical Proof Systems for Logic II, Consensus Programs and Their Processing (Extended Abstract)

Helena Rasiowa[1] * and V. Wiktor Marek[2] **

[1] Institute of Mathematics, Warsaw University, Warsaw, Poland
[2] Computer Science Department, University of Kentucky, Lexington, KY 40506-0027

Abstract. We continue the investigations of [Ra90, Ra91, RM89] and study the automated theorem proving for reasoning about perception of reasoning agents and their consensus reaching. Using the techniques of [Ra91] and of Logic programming ([Ap90, NS93]) we develop the processing techniques for consensus programs.

1 Introduction

Investigations concerning a systematic logical approach to reasoning about knowledge of one or several intelligent agents have in recent years been developed by logicians and computer scientists. These investigations lead to a variety of different logical systems based on various paradigms. We mention here the work of Halpern and Moses ([HM84]), Fagin, Halpern, and Vardi [FHV90]), Mazer ([Ma88]), Orlowska ([Or90]) just to indicate to the reader that the issues of knowledge in the distributive environment are studied widely. TARK proceedings ([Ha86, Va, Pa90, Mo92]) include numerous papers devoted to the subject.

The authors ([RM89]) proposed an approach to reasoning about the perception and the knowledge of groups of fully communicating agents based on the point of view expressed in these principles:

1. The sharpness of agent's perception depends on agent's abilities.
2. Abilities of various agents may be comparable or not comparable (in the sense that one agent may be more capable in one situation, whereas other agent may be more capable in another situation).
3. Agent's knowledge about a reality is only approximate.
4. Agent's knowledge about a predicate (property) p can be reflected by her perception of p, that is the characteristic features of p acquired in a process of collecting information, conducting research, etc.
5. Agent's knowledge about a property p can be reflected in her abilities to recognize various features and attributes of objects. This may require access to specific recognition medium such as a database, laboratory, test etc.

* Work partially supported by Polish Government grant KBN 2 2051 91 02. E-mail: hrasiowa@mimuw.edu.pl

** Work partially supported by the U.S. National Science Foundation grant IRI-9012902. E-mail: marek@cs.uky.edu

Starting with this intuition the authors ([RM89]) developed a logic with two types of connectives: perception connectives and knowledge connectives. Perception connectives correspond to the approximation connectives of predicate logic ([Ra87, Ra88, Ra90]). The knowledge connectives correspond to those of the logic of knowledge of Orlowska ([Or90]). The logics without knowledge operators, but with the perception connectives only, have been reexamined by Rasiowa ([Ra91]), under the terms of perception logics. Rasiowa ([Ra91]) established an automated theorem proving technique for such logics. A similar approach has been proposed by Fitting ([Fi92]).

In this paper we look again at the perception logics (that is we leave the knowledge part of the language unattended) and prove several results relating the ordinary resolution method of Robinson ([Rc65]) and the form of the resolution discussed in [Ra91].

Our approach here is based on a different technique, much more closely connected to the current presentations of resolution. In this we follow the current texts by Apt ([Ap90]) and Nerode and Shore ([NS93]). The technique used in [Ra91] employed a variant of an argument used by Orlowska in her investigations on the resolution for multivalued logics.

The space restrictions force us to eliminate the proofs of the results of this paper. We also had to eliminate section dealing with the *lifting of our results* to the predicate calculus case. These results will be included in the full journal presentation.

2 Perception logics and resolution

Let T be a finite set (of reasoning agents). The set T is endowed with a partial ordering $\leq_T$. Intuitively, $s \leq_T t$ means that the agent t is more perceptive that the agent s. This is interpreted as follows. When both the agents s and t are asked about a specific fact p, the agent t will be less gullible. Her abilities are better in recognizing if the fact p really happens. Specifically, t can find that p did *not*, actually, happen (whereas s perceives p as true). In particular, when $s \leq_T t$ and p is an atomic statement then, whenever t perceives p to be true, s perceives p to be true. In other words, the property $s \leq_T t$ means that this sharper perception happens for all possible facts p. Notice, that a similar property is called "dominance" in Fitting ([Fi92]). All agents observe the same reality. This means that each agent is endowed with a valuation of the set of atoms At (in the propositional case) or a relational structure (with the same underlying algebra, that is with the same objects, and the same interpretation of the function symbols and the constants) - in the predicate case.

Denoting by V_t the valuation assigned to the agent t the requirement of better perception for a stronger agent is formally expressed as

$$w \leq_T t \quad \Rightarrow \quad \forall_{p \in At}\ V_t(p) \leq V_w(p) \tag{1}$$

In the predicate case, denoting by $\mathcal{A}_t$ the relational system assigned as the

perception of the agent t we have, for every predicate letter p,

$$w \leq_t t \quad \Rightarrow \quad p^{\mathcal{A}_t} \subseteq p^{\mathcal{A}_w} \tag{2}$$

Formally, the language of perception logic is the language of the classical logic extended by unary modal operators d_t, for $t \in T$. Intuitively, $d_t\varphi$ means that the agent t (and, as will turn out all the agents with weaker perception) perceives φ.

The semantics for our language is determined by the consensus interpretation. In the predicate case a T- *reality* for the underlying language is a collection of first order relational structures. The collection is indexed by the set T. All the structures in a T-reality have the same underlying algebra and the T-reality must satisfy Condition (2).

Analogously we define the notion of T-*reality* for a propositional case. It is a collection of valuations of the underlying set of atoms. Moreover we require (1).

Once it is clear what we mean by the realities for our language, we define the notion of satisfaction. The fact that a T-reality $\mathcal{M} = \langle \mathcal{A}_t \rangle_{t \in T}$ satisfies φ means that the consensus about φ has been reached. This, of course implies that the set of formulas true in a T-reality is *not* complete. The notion of satisfaction for formulas is defined in a roundabout way. We give the full definition of satisfaction for the predicate case. The propositional case can be easily described by an obvious modification of the clause (a) and eliminating quantifier cases (g) and (h). First we define the relation $\mathcal{M} \models d_t(\varphi)[v]$ (where v is a valuation of variables).

(a) $\mathcal{M} \models d_t(p_i(x_1, \ldots, x_m))[v]$ iff $\mathcal{A}_t \models p_i(x_1, \ldots, x_m)[v]$
(b) $\mathcal{M} \models d_t(\phi \vee \psi)[v]$ iff $\mathcal{M} \models d_t(\phi)[v]$ or $\mathcal{M} \models d_t(\psi)[v]$
(c) $\mathcal{M} \models d_t(\phi \wedge \psi)[v]$ iff $\mathcal{M} \models d_t(\phi)[v]$ and $\mathcal{M} \models d_t(\psi)[v]$
(d) $\mathcal{M} \models d_t(\phi \Rightarrow \psi)[v]$ iff for all $s \leq_T t$, $\mathcal{M} \models d_s(\phi)[v]$ implies $\mathcal{M} \models d_s(\psi)[v]$
(e) $\mathcal{M} \models d_t(\neg\phi)[v]$ iff for all $s \leq_T t$, not $(\mathcal{M} \models d_s(\phi)[v])$
(f) $\mathcal{M} \models d_s(d_t(\phi))[v]$ iff $\mathcal{M} \models d_t(\phi)[v]$
(g) $\mathcal{M} \models d_t(\forall x_i \phi)[v]$ iff for all $a \in M$, $\mathcal{M} \models d_t(\phi)[v(i/a)]$
(h) $\mathcal{M} \models d_t(\exists x_i \phi)[v]$ iff there exists $a \in M$, $\mathcal{M} \models d_t(\phi)[v(i/a)]$

Next, we define the satisfaction for all formulas of the underlying language L.

$$\mathcal{M} \models \Psi[v] \text{ if and only if } \mathcal{M} \models d_t(\Psi)[v] \text{ for all } t \in T.$$

A complete axiomatization for the relation $\models$ has been given in [RM89].

Here is a short list of fundamental properties of the satisfaction (consensus) relation for a T-reality $\mathcal{M}$.

$$\mathcal{M} \models \neg d_t(\varphi)[v] \text{ iff } \textit{not } \mathcal{M} \models d_t(\varphi)[v] \tag{3}$$

$$\mathcal{M} \models d_w(\neg d_t(\varphi))[v] \text{ iff } \mathcal{M} \models \neg d_t(\varphi)[v] \tag{4}$$

$$\mathcal{M} \models (d_t(\varphi) \Rightarrow d_t(\psi))[v] \text{ iff } \textit{not} \mathcal{M} \models d_t(\varphi)[v] \textit{ or } \mathcal{M} \models d_t(\psi)[v] \tag{5}$$

$$\forall_{w,t \in t} w \leq_T t, \mathcal{M} \models d_t(\varphi)[v] \textit{ implies } \mathcal{M} \models d_w(\varphi)[v] \tag{6}$$

$$\forall_{w,t\in T}\, w \leq_T t \quad \textit{implies} \quad \mathcal{M} \models (d_t(\varphi) \Rightarrow d_w(\varphi))[r] \tag{7}$$

Let us introduce the notion of *D-atom*. It is a formula of the form $d_t p(s_1, \ldots, s_r)$ where $t \in T$ and $s_1, \ldots, s_r$ are terms of the language (in the propositional case no term is present). A *D-literal* is an D-atom or its negation. A *D-clause* is a disjunction of D-literals.

Summing up all properties of formulas of the form $d_t\varphi$, we quote the following result of [Ra91]

Proposition 1. *For every formula of the form $d_t\varphi$ there is a finite set of D-clauses S such that $d_t\varphi$ and S are equisatisfiable. That is there is a T-reality $\mathcal{M}$ satisfying $d_t\varphi$ if and only if there is a T-reality $\mathcal{M}'$ such that $\mathcal{M}'$ satisfies all the D-clauses from S.*

Proposition suggests that we may be facing a situation similar to that encountered in automated theorem proving. A version of resolution principle may work here. Indeed it is the case. In [Ra91] Rasiowa found a variant of the resolution principle suitable for our context. We will refer to this rule as T-resolution rule. We shall denote this rule by res_T.

$$\frac{A' \cup \{d_t(p(t_1, \ldots t_k))\}, \qquad A'' \cup \{\neg d_w(p(s_1, \ldots s_k))\}}{(A' \cup A'')\Theta} \quad \text{providing } w \leq_T t \tag{8}$$

Notice the asymmetry of the rule res_T. Its applicability is restricted by the condition $w \leq_T t$. Here we assume that the parent clauses are standardized apart. Θ is a most general unifier of atoms $p(t_1, \ldots t_k)$ and $p(s_1, \ldots s_k)$.

As usual, by Herbrand T-reality we mean a T-reality whose underlying universe consists of ground terms of the language.

The following result is proved in [Ra91]

Proposition 2. *Let S be a set of D-clauses. Then there is a T-reality satisfying S if and only if the closure of S under T-resolution does not contain an empty clause.*

3 Automated theorem proving for consensus reaching

We will prove a basic result on the relationship of the asymmetric resolution rule introduced in [Ra91] and the usual resolution rule (for best description see Nerode and Shore [NS93]).

Let $\langle T, \leq_T \rangle$ be a poset, and let At be a set of atoms. Recall that a D-atom is an expression of the form $d_t(p)$ where $p \in At$, $t \in T$. Similarly, a D-literal is a D- atom or its negation. Next, a D-clause is a finite set of D- literals. As usual, $\Box$ is the empty clause (interpreted as falsity).

In our context, the T-resolution rule is the following rule of proof:

$$res_T \qquad \frac{A' \cup \{d_t(p)\}, \quad A'' \cup \{\neg d_w(p)\}}{A' \cup A''}$$

where $w \leq_T t$.

The ordinary resolution rule in our setting takes this form:

$$res \qquad \frac{A' \cup \{d_t(p)\}, \qquad A'' \cup \{\neg d_t(p)\}}{A' \cup A''}$$

The T-resolution rule is asymmetric. We can resolve on a D-atom $d_t(p)$ against $\neg d_w(p)$ only if $w \leq_T t$. Hence, D-clauses $\{d_w(p)\}$ and $\{\neg d_t(p)\}$ ($w \leq_T t$) do not entail $\Box$. This agrees with our paradigm; since $w \leq_T t$, the atom p may be perceived by the agent w as true and the agent t as false without the contradiction.

Let S be a set of D-clauses. By $\mathcal{R}_T(S)$ we mean the closure of the set S under the rule res_T. Similarly, $\mathcal{R}(S)$ is the closure of S under the usual resolution rule. Notice that res_T is a generalization of the ordinary resolution rule. Indeed, every derivation using the ordinary resolution *is* a valid res_T reasoning. This is because the relation $\leq_T$ is reflexive. As noticed above, the converse does not necessarily hold. That is, a refutation using T-resolution does not need to be a resolution refutation.

Denote by D_T, the *diagram of* T, the following set of D-clauses:

$$D_T = \{\{\neg d_t(p) \vee d_w(p)\} : \qquad w \leq_T t,\ p \in At\}$$

A more intuitive representation for D_T is:

$$D_T = \{d_t(p) \Rightarrow d_w(p) : \qquad w \leq_T t, p \in At\}$$

The set D_T codifies our knowledge about the relationship between the agents. If the agent t is more perceptive than the agent w (which in our system is encoded by $w \leq_T t$), and t accepts a fact p then, certainly, w accepts the fact p - but not vice versa.

We have now the basic result on the connection between the ordinary resolution rule and the T-resolution rule. This result is fundamental for the rest of the paper. Once we prove it, we will be able to list the most of the results of ordinary automated theorem proving with resolution to the case of D-clauses and T-resolution.

Theorem 3. *Let S be a set of D-clauses. Then $\Box \in \mathcal{R}_T(S)$ if and only if $\Box \in \mathcal{R}(S \cup D_T)$.*

Proposition 4. *Let S be a set of D-clauses. Then there is a T-reality satisfying S if and only if $\Box \notin \mathcal{R}(S \cup D_T)$.*

Corollary 5 [Ra91]. *Let S be a set of D-clauses. Then there is a T-reality satisfying S if and only if $\Box \notin \mathcal{R}_T(S)$.*

Let us restrict now to the case of Horn D-clauses. Here, the important observation is that all the D-clauses in the diagram of T, D_T, are Horn D-clauses.

Recall that a Horn D-clause is a D-clause that contains at most one positive literal. In our setting Horn D-clauses are of the form:

$$\{d_s(p), \neg d_{w_1}(q_1), \ldots, \neg d_{w_k}(q_k)\}$$

or of the form:

$$\{\neg d_{w_1}(q_1), \ldots, \neg d_{w_k}(q_k)\}$$

The D-clause $\{d_s(p), \neg d_{w_1}(q_1), \ldots, \neg d_{w_k}(q_k)\}$ is called a T-program clause and usually denoted by

$$d_s(p) \leftarrow d_{w_1}(q_1), \ldots, d_{w_k}(q_k)$$

The second type of Horn D-clause is called a *goal* and is denoted in the logic programming by:

$$\leftarrow d_{w_1}(q_1), \ldots, d_{w_k}(q_k)$$

For the notion of linear input resolution see Nerode and Shore [NS93]. We can consider the linear input resolution in our context. That is, we consider a linear tree starting with a goal and applying T-resolution instead of ordinary resolution.

The crucial observation now is that not only D_T consists of Horn clauses, but the transformations described in both parts of out Theorem 3 preserve the linear input resolution.

Specifically we have:

Theorem 6. *Let S be a set of Horn D-clauses. Then there is no T-reality satisfying S if and only if S possesses a linear input refutation using the T-resolution rule.*

Corollary 7. *Let S be a theory consisting of Horn D-clauses. The following are equivalent:*

1. *S possesses a refutation using T-resolution rule.*
2. *S possesses a linear input refutation using T-resolution rule.*
3. *$S \cup D_T$ possesses a linear input refutation using ordinary resolution rule.*
4. *$S \cup D_T$ possesses a refutation using ordinary resolution rule.*
5. *$S \cup D_T$ is unsatisfiable.*
6. *There is no T-reality satisfying S.*

4 Consensus Programs and their processing

In this section we discuss T-programs. Recall from Section 3 that a T-program clause is a D-clause of the form

$$C = d_s(p) \leftarrow d_{w_1}(q_1), \ldots, d_{w_k}(q_k)$$

A *T-program* is any set of T-program clauses. Intuitively, a T-program P describes a T-reality, with various interconnections of agent perceptions. Intuitively, the T program clause C tells us that in the T-reality described by P

this happens: whenever the agent w_1 perceives the fact q_1, and the agent w_2 perceives the fact q_2 etc. then the agent s perceives p.

The valuations of D-atoms can be put into a one-to-one correspondence with the subsets of D-atoms in the usual fashion. Therefore we will not distinguish between the valuations of D-atoms and sets of D-atoms. Also, it is obvious that T-realities are naturally ordered by inclusion.

Proposition 8. *For every T-program P there exists the least T-reality satisfying P.*

We will describe now a sound and complete method of processing queries to T-programs.

The way we are going to process our queries is a variant of the usual processing of logic program and will reflect precisely the difference between the usual resolution rule and the T-resolution rule. To set a terminology, we shall call this operation *T-matching*. Formally, a D-atom $d_s(p)$ *T-matches* a T program clause $C = d_t(p) \leftarrow d_{u_1}(q_1), \ldots, d_{u_k}(q_k)$ if $s \leq_T t$. It should be clear that when $s = t$ then matching reduces to selection of an D-atom $d_t(p)$ for expansion.

The second part of the processing procedure coincides with that of ordinary logic programming. Once we have a goal $\leftarrow d_{w_1}(q_1), \ldots, d_{w_k}(q_k)$ and select within it a D-atom $d_{u_i}(q_i)$ and the D-atom $d_{u_i}(q_i)$ T-matches a T program clause

$$d_t(q_i) \leftarrow d_{z_1}(h_1), \ldots, d_{z_n}(h_n)$$

then we create a new goal

$$\leftarrow d_{w_1}(q_1), \ldots, d_{w_{i-1}}(q_{i-1}), d_{z_1}(h_1), \ldots, d_{z_n}(h_n), d_{w_{i+1}}(q_{i+1}), \ldots d_{w_k}(q_k)$$

this process is called *expansion*. Hence, the expansion process corresponds to one application of T-resolution between the current goal and some T program clause in P.

Let G be a goal. We say that *the goal G succeeds* if there is a sequence of goals $G_0, \ldots, G_m$ such that $G_0 = G$, $G_m = \square$ and each G_{i+1} arises from G_i by selecting a D-atom in G_i, matching it with some T program clause in P and expanding.

We now have the following result.

Proposition 9. *Let P be a T-program. Let G be a goal. Then there is no T-reality satisfying $P \cup \{G\}$ if and only if the goal G succeeds.*

Let us look at an example.

Example 1. Let T be the partial ordering of Figure 3. Consider the following simple T-program

$d_u(p) \leftarrow d_u(q), d_s(r)$
$d_t(r) \leftarrow$
$d_s(q) \leftarrow$

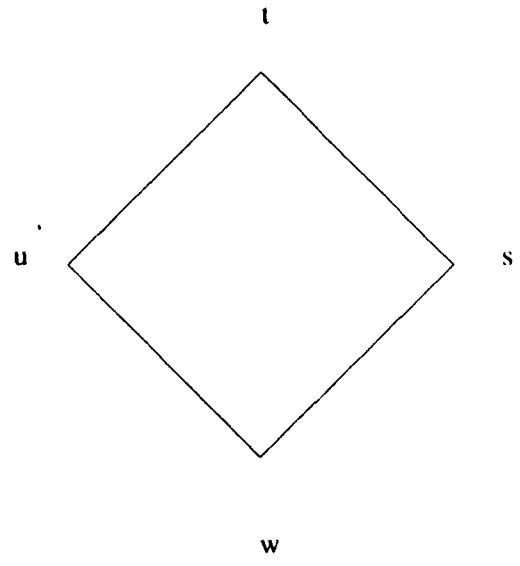


Fig. 1. Partial Ordering T

Given the goal $\leftarrow d_u(p)$ we notice that the D-atom in that goal T-matches the first clause in our program. It is not identical with it, just T-matches it, because $w \leq_T u$. This creates by expansion a new goal $\leftarrow d_u(q), d_s(r)$. The first D-atom in this goal matches the second clause in our program because $w \leq t$. The expansion creates now the goal $\leftarrow d_s(r)$. The D-atom in this goal matches the third clause. The result of expansion is now $\Box$ and so the original goal succeeds.

The completeness result (Proposition 9) implies the following proposition.

Proposition 10. *Let $d_s(p)$ be a D-atom. Let P be a T-program. Then the goal $\leftarrow d_s(p)$ succeeds if and only if $d_s(p)$ belongs to the least T-reality satisfying P.*

The operator T_P associated to logic program can be lifted to the present situation with some modifications. The difference is that as we compute new D-atoms, we also need to add D-atoms perceived by less perceptive agents.

Specifically, given a T-program P define an operator S_P as the T-closure of the set

$$\{d_u(p) : \text{ There exists } C \in P, C = d_s(p) \leftarrow d_{u_1}(q_1), \ldots, d_{u_k}(q_k), \text{ and } d_{w_1}(q_1) \in M, \ldots, d_{u_k}(q_k) \in M, w \leq_T s\}.$$

Proposition 11. *1. The operator S_P is monotone and finitizable.*
2. Hence, S_P possesses the least fixpoint which is equal to $\bigcup_{n \in \omega} S_P^n(\emptyset)$.

Similarly to the case of ordinary logic programs we have the following theorem.

Theorem 12. *Let P be a T-program. Then*

1. *The least T-reality satisfying P coincides with the least fixpoint of S_P.*
2. *The least fixpoint of S_P coincides with the set of D-atoms $d_s(p)$ for which the goal $\leftarrow d_s(p)$ succeeds.*

4.1 Processing the consensus queries

Now, it is clear how we can get a result about processing atomic queries *not* involving the operator d_t for T-programs. Such query to a program P is a query about *consensus* in the T-reality described by the program P. To get the answer to such query, say p (where p) is an atom, we must check if *all* D-atoms $d_t(p)$ succeed. It turns out that we do not need to check all atoms $d_t(p)$. It follows immediately from the basic property (7) of T-realities (see Section 2) that it is enough to check if the queries $d_t(p)$ succeed for all the *maximal* elements t of T. Similarly, if a query $d_u(p)$ fails for every *minimal* element s of T then the T-reality described by our program P satisfies $\neg p$.

At a bigger cost we can now process an arbitrary consensus query. Given a propositional formula φ, the formula φ is satisfied in the least T-reality $\mathcal{M}_T$ satisfying the program P if and only if all formulas $d_t(\varphi)$ are satisfied in $\mathcal{M}_T$. Assume for a moment that φ does not contain the implication functor, and that the negation functor appears only in front of expression of the form $d_s(\psi)$. Clearly, the formula $d_t(\varphi)$ is logically equivalent to a set of D-clauses. Thus we need to be able to check whether a D-clause is satisfied in $\mathcal{M}_T$. But such D-clause C is satisfied in $\mathcal{M}_T$ if and only if one of D-literals in C is satisfied in $\mathcal{M}_T$. This, together with the above remarks on testing the validity of D-literals in $\mathcal{M}_T$, gives a method for testing consensus for *arbitrary* propositional formulas.

Finally, if φ does contain implication and unrestricted negation, then, again, we can test the consensus about φ, but now the cost is bigger. This happens because in the recursive definition of satisfaction we need to consult the perception of less perceptive agents.

References

[Ap90] K. Apt (1990), "Logic Programming", In: *Handbook of Theoretical Computer Science*, J. van Leeuven ed., pp. 493-574, MIT Press, Cambridge, MA.

[FHV90] Fagin, R., Halpern, J. Y., Vardi, M. (1988), "Model-theoretical Analysis of Knowledge", IBM Research report RJ 6461.

[Fi92] Fitting, M. (1992), Many-valued Modal Logics II, *Fundamenta Informaticae* 17, pp. 55-73.

[Ha86] Halpern, J. Y. (1986) (ed), *Theoretical Aspects of Reasoning About Knowledge*, Morgan Kaufmann.

[HM84] Halpern, J.Y., Moses, Y. (1984), "Toward a Theory of Knowledge and Ignorance: Preliminary Report", *Proceedings of AAAI Workshop on Non-Monotonic Reasoning*, pp. 125-143.

[Ma88] Mazer, M.S. (1988), "A Knowledge Theoretic Account of Recovery in Distributed Systems", *TARK '88*, M. Vardi ed., pp. 309-324.

[Mo92] Moses, Y., (1992) (ed), *Theoretical Aspects of Reasoning About Knowledge*, Morgan Kaufmann.

[NS93] Nerode, A., Shore, R. (1993), *Logic for Applications*, Springer-Verlag.

[Or85] Oriowska, E. (1985), "Mechanical proof methods for Post logics", Logique Anal. N.S., 28 Annee, 110-111, 173-192.

[Or90] Orlowska, E. (1990), "Logic for Reasoning about Knowledge", Z. Math. Logik Grund. Math.

[OS86] Orlowska, E., Sanders, J. (1986), "Knowledge Transfer in Distributed Systems", unpublished manuscript.

[Pa90] Parikh, R. (1990) (ed), *Theoretical Aspects of Reasoning About Knowledge*, Morgan Kaufmann, 1990.

[Pa82] Pawlak, Z. (1982), "Rough Sets", *International Journal of Computer And Information Sciences* 11, pp. 341-356.

[Ra86] Rasiowa, H. (1986), "Rough Concepts and ω^+ valued Logic", *Proceedings ISMVL '86*. IEEE Press, pp. 282-289.

[Ra87] Rasiowa, H. (1987), "Algebraic Approach to Some Approximate Reasonings", *Proceedings ISMVL '87*. IEEE Press, pp. 342-347.

[Ra88] Rasiowa, H. (1988), "Logic of Approximation Reasoning", *Proceedings of CSL '87*. Springer LN in Computer Science 329, pp. 188-210.

[Ra90] Rasiowa, H. (1990), "On approximation logics: A survey", Kurt Gödel Gesellschaft, Jahrbuch 1990, pp. 63-87.

[Ra91] Rasiowa, H. (1991), "Mechanical Proof Systems for Logic: Reaching Consensus by Groups of Intelligent Agents", *International Journal of Approximate Reasoning* 5, pp. 415-432.

[RM89] Rasiowa, H., Marek, W. (1989), "On reaching consensus by groups of intelligent agents", *Proceedings ISMIS'89*, pp. 234-243, North-Holland.

[RS63] Rasiowa, H., Sikorski, R. (1963), "The Mathematics of Metamathematics", PWN, Warszawa, (3rd ed. 1970).

[Ro65] Robinson, G.A. (1965), "A machine oriented logic based on the resolution principle", *Journal of the Association for Computing Machinery* 12, pp. 23-41.

[Va] Vardi, M. Y. (1988), (ed), *Theoretical Aspects of Reasoning About Knowledge*, Morgan Kaufmann.

The Logic of Only Knowing as a Unified Framework for Non-monotonic Reasoning

Jianhua Chen

Computer Science Department
Louisiana State University
Baton Rouge, LA 70803, USA
E-mail: jianhua@bit.csc.lsu.edu

Abstract

We propose to use the logic of *only knowing* (OL) by Levesque [6] as a unified framework that encompasses various non-monotonic formalisms and logic programming. OL is a modal logic which can be used to formalize an agent's introspective reasoning and to answer epistemic queries to databases. The OL logic allows one to formally express the statement "α is *all* I know" (in symbols, Oα) and to perform inferencing based on only-knowing, which is very useful for commonsense reasoning. Another nice thing about the OL logic is that it has a clear model-theoretic semantics and a simple proof theory, which is sound for the quantificational case, and both sound and complete for the propositional case.

We establish the relations between OL and various non-monotonic logics (such as default logic, circumscription) and logic programming, thus extending the existing works relating the OL logic with other non-monotonic reasoning formalisms (e.g., Levesque showed [6] that autoepistemic logic can be embeded in OL). This is accomplished by finding the connection between OL and MBNF, the logic of *Minimal Belief and Negation as Failure* proposed by Lifschitz [8, 9], which is known to have close relationship with logic programming and other non-monotonic logics. Our results show that OL can be used as a unified framework to compare different non-monotonic formalisms based on the same domain.

1. Introduction

In this paper, we investigate the relationship between OL, the logic of only knowing, and default logic, circumscription, and several logic programming languages. We show that circumscription and a substantial class of default logic can be embeded in OL, and that normal logic programs and extended logic programs (with

classical negation) are also included in OL. This is done by connecting OL with the MBNF logic which is known to be a general framework for non-monotonic reasoning. The results in this paper, coupled with existing work relating OL with autoepistemic logic and using OL to perform epistemic queries, make the case for OL as a unified framework for non-monotonic reasoning.

The logic of only knowing is proposed by Levesque [6]. The motivation for the OL logic is to show that some patterns of non-monotonic reasoning can be captured by using the classical notions of logic (satisfiability, validity, implication). To achieve that, Levesque extends the classical modal logic by formalizing the notion of "only knowing". He argued that in modeling an agent's reasoning about its own belief and knowledge, we need to formalize not only the notion that "ϕ is believed" (in symbols, $\mathbf{B}\phi$), but also the notion that "ϕ is *all* that is believed" (in symbols, $\mathbf{O}\phi$). Thus the OL logic has two modal operators "**B**" and "**O**". In [6], a model-theoretic semantics is defined for OL and a simple proof theory is established. The proof theory is essentially an extension of proof theory for the weak S5 modal logic (also called K45). It is both sound and complete for the propositional case, and sound for the quantificational case. The nice thing about the proof theory is that with little more than standard modal logic, we can perform inferencing in OL and epistemic query-answering for non-monotonic databases. It is shown that the notion of only knowing corresponds exactly to the notion of stable expansion in autoepistemic logic (AE logic) by Moore [15] and thus autoepistemic logic can be embeded in OL.

Obviously, with the capability to express the notion of "only knowing", and a clear model-theoretic semantics, as well as a simple proof theory, the OL logic is a very attractive candidate to be a general framework for various forms of non-monotonic reasoning. However, in spite of the previous works on clarifying the relationship between OL logic and other non-monotonic formalisms [4, 6, 10, 12], the picture is still not very clear as to how OL relates with other logics and logic programming languages. In this paper, we attempt to address this issue and thus establish the link between OL and various non-monotonic logics and logic programming. This work will enhance our understanding about the nature of various non-monotonicity captured by different formalisms and it will establish OL logic as a unified framework for non-monotonic reasoning. The OL logic formulas we discuss in this paper are essentially AE logic formulas, and thus this work also clarifies the relationship between AE and other non-monotonic formalisms. As will be seen in Section 4, our approach to the relationship between AE logic and other formalisms (such as default logic and circumscription) is different from the approaches of Konolige [4], Marek and Truszczynski [12], and Lifschitz [10].

There are many recent works relating both AE logic and default logic to various forms of logic programming [2, 3, 16, 18]. Some recent works (e.g, [18]) seem to suggest that AE logic is not suitable for formalizing logic programs with classical negations whose semantics extends the stable model semantics of normal program and that default logic is *the* right one to do such job. However, we will show in this

paper, that the OL logic (and hence the AE logic) *does include* the extended logic programs under a simple translation, therefore confirming the suitability of using AE logic to formalize extended forms of logic programming. This result partially overlaps with the more recent work of Lifschitz and Schwarz [11], and that of Marek and Truszczynski [13].

Our approach to the connections between OL and other non-monotonic logics is to relate the OL logic with MBNF, the logic of minimal belief and negation as failure by Lifschitz [8, 9]. Lifschitz has shown that MBNF can be used as a general logical framework in the sense that various non-monotonic formalisms such as default logic, circumscription, as well as logic programming, can be embeded in MBNF, and that epistemic query answering can also be cast in the framework of MBNF. In [1], we show that a substantial subclass of MBNF theories can be embeded in OL, which essentially forms the foundation for this paper. The advantage of using OL (instead of MBNF) as a general logical framework is that OL has a simple proof theory while MBNF does not have one.

This paper is organized as follows. In section 2, we will give basic definitions and briefly review OL and MBNF. We briefly restate the result from [1] regarding the OL and MBNF connection in Section 3. The main results will be presented in section 4, in which we establish the relations between OL and default logic, circumscription, and various logic programming systems. In this report, we focus on only the propositional logics (OL, MBNF, default, logic programming, etc.), and we leave the quantificational for future work.

2. Preliminaries

We give the basic definitions and briefly review OL and MBNF in this section. The reader is referred to [6, 8, 9] for detailed discussion about these two logics. In both OL and MBNF, we will deal with propositional languages extended by adding some modal operators ("**B**" and "**O**" in OL, "**B**" and "not" in MBNF). A theory is a finite set of formulas (axioms). A formula is called *objective* if it does not include any modal operators; it is called *subjective* if each occurrence of objective formula is within the scope of a modal operator. Objective theories and subjective theories are defined similarly.

The structures which define the truth/falsity of a formula ϕ in a modal language will be different from those used in defining the truth value of ordinary propositional formulas. Consider a theory Δ in either OL or MBNF and assume $\{p_1, p_2, ..., p_n\}$ are all the proposition symbols occurring in Δ. An *interpretation* I is a set of atoms (propositions) from $\{p_1, p_2, ..., p_n\}$. We denote the set of all such interpretations as Ω. Clearly Ω has the cardinality 2^n. A *structure* is of the form $\langle I, S\rangle$ where $I \in \Omega$ is an interpretation and $S \subseteq \Omega$ is a set of interpretations. Intuitively, I represents the "real world" and S represents the set of "possible worlds" accessible from I. Notice that I need not be a member of S. Essentially, the truth value of a modal formula ϕ

will be determined at each structure ⟨I, S⟩. Let Π be the set of all structures and define the order relations "<" and "≤" over Π as follows. Let $\langle I_1, S_1\rangle$, $\langle I_2, S_2\rangle$ be structures in Π, define $\langle I_1, S_1\rangle < \langle I_2, S_2\rangle$ if $S_1 \subset S_2$, define $\langle I_1, S_1\rangle \leq \langle I_2, S_2\rangle$ if $S_1 \subseteq S_2$. The logics MBNF and OL will focus on the *maximal* structures which satisfy a theory in such logics.

2.1 The logic of Only Knowing

Levesque developed [6] a modal logic to formulate and infer about an agent's knowledge and belief. We call it the logic of *only knowing* (OL). A propositional OL language is obtained by adding two modal operators **B** and **O** to a propositional language. An OL theory is called *basic* if it does not contain any occurrence of the **O** operator. Given a structure ⟨I, S⟩ and formulas ϕ and ψ in OL, we have the following truth value definitions:

(1) If ϕ is an atom, then ϕ is true in ⟨I, S⟩ if and only if $\phi \in$ I.

(2) $\neg\phi$ is true in ⟨I, S⟩ if and only if ϕ is *not* true in ⟨I, S⟩.

(3) $\phi \wedge \psi$ is true in ⟨I, S⟩ if and only if both ϕ and ψ are true in ⟨I, S⟩.

(4) $\mathbf{B}\phi$ is true in ⟨I, S⟩ if and only if for each J ∈ S, ϕ is true in ⟨J, S⟩.

(5) $\mathbf{O}\phi$ is true in ⟨I, S⟩ if and only if $\mathbf{B}\phi$ is true in ⟨I, S⟩ and for any J ∈ Ω, ϕ being true in ⟨J, S⟩ will imply J ∈ S.

It is clear that for a structure ⟨I, S⟩, the truth value of an objective formula does not depend on S and the truth value of a subjective formula does not depend on I. We say ⟨I, S⟩ satisfies a formula ϕ if ϕ is true in ⟨I, S⟩. A structure ⟨I, S⟩ is a *model* of a theory Δ if every formula in Δ is satisfied (true) in ⟨I, S⟩. Later on we can see that the notion of "only knowing" ($\mathbf{O}\phi$) in OL is very much similar to the notion of *minimal belief* ($\mathbf{B}\phi$) in MBNF. Intuitively, to say ⟨I, S⟩ satisfies $\mathbf{O}\phi$ means that ⟨I, S⟩ satisfies $\mathbf{B}\phi$ and that S is a *maximal* set of interpretations satisfying $\mathbf{B}\phi$.

2.2 Minimal Belief and Negation as Failure

Lifschitz proposed the logic of *minimal belief and negation as failure* (MBNF) [8, 9]. Here the presentation conforms to [9]. An MBNF language is formed by adding the modal operators "**B**" and "not" to a propositional language. For example, $\Delta = \{not(p) \rightarrow q, \mathbf{B}r \vee \mathbf{B}(s \vee not(q))\}$ is a theory in MBNF. A formula or a theory is called *positive* if it does not contain any occurrence of the "*not*" operator.

The definition of a *positive* formula ϕ being true in a structure ⟨I, S⟩ (i.e., ⟨I, S⟩ satisfies ϕ) is essentially the same as the above (1) - (4) in OL logic. ⟨I, S⟩ is said to satisfy a positive theory Δ if every formula in Δ is true in ⟨I, S⟩. However, ⟨I, S⟩ is not necessarily a *model* of Δ, even if ⟨I, S⟩ satisfies Δ. To be a model of a positive theory Δ, ⟨I, S⟩ has to be a *maximal* structure which satisfies Δ.

To define the notion of a model for a general theory Δ in MBNF, a triple of the form $\langle I, S_b, S_n \rangle$ is used, where $I \in \Omega$, $S_b \subseteq \Omega$ and $S_n \subseteq \Omega$, S_b denotes the set of possible worlds used to determine the belief (formulas of the form $\mathbf{B}\phi$) and S_n represents the set of worlds used to determine the negation as failure (formulas of the form not(ϕ)). To be more specific, given a triple $\langle I, S_b, S_n \rangle$, and MBNF formulas ϕ and ψ, the definitions of ϕ, $\neg\phi$, $\phi \wedge \psi$ and $\mathbf{B}\phi$ being true in $\langle I, S_b, S_n \rangle$ are parallel to the (1) - (4) for the case of positive theory, with $\langle I, S_b, S_n \rangle$ in place of $\langle I, S \rangle$ and S_b in place of S. In addition, not(ϕ) is true in $\langle I, S_b, S_n \rangle$ if and only if there is $J \in S_n$ such that $\neg\phi$ is true in $\langle J, S_b, S_n \rangle$.

For a given theory Δ and a given set of interpretations $S \subseteq \Omega$, define $\Gamma(\Delta, S)$ to be the set of all *maximal* structures $\langle I, S' \rangle$ such that every formula in Δ is true in $\langle I, S', S \rangle$. A structure $\langle I, S \rangle$ is a *model* of Δ if $\langle I, S \rangle \in \Gamma(\Delta, S)$, i.e., if $\langle I, S, S \rangle$ satisfies Δ, and there is no proper superset S' of S such that $\langle J, S', S \rangle$ satisfies Δ for any $J \in \Omega$.

For an objective formula ϕ, we use Mod(ϕ) to denote the set of interpretations which (propositionally) satisfy ϕ, i.e., Mod(ϕ) is the set of propositional models of ϕ. Consider the positive theory $\Delta_1 = \{\mathbf{B}p \vee \mathbf{B}q, \mathbf{B}(r \vee s)\}$. It is not difficult to see that the models of Δ_1 are of the form $\langle I, \text{Mod}(p) \cap \text{Mod}(r \vee s) \rangle$ and $\langle I, \text{Mod}(q) \cap \text{Mod}(r \vee s) \rangle$, where I is any interpretation. For the theory $\Delta_2 = \{\text{not}(p) \rightarrow q\}$, the models of Δ_2 are of the form $\langle I, \Omega \rangle$ for any $I \in \text{Mod}(q)$. The theory $\Delta_3 = \{\neg \text{not}(p)\}$ has no models.

3. Between OL and MBNF

In this section, we restate the results in [1] connecting OL with MBNF. Given an MBNF or OL theory Δ which is a finite set of formulas, we use Δ to denote both the set and the conjunction of the formulas in the set. Thus we can talk about the formula (theory) $\mathbf{B}\Delta = \{\mathbf{B}\phi \mid \phi \in \Delta\}$ in MBNF or OL. Similarly, for an OL theory Δ, we use $\mathbf{O}\Delta$ to denote the formula $\mathbf{O}\Psi$, where Ψ is the conjunction of all formulas in Δ. Note that $\mathbf{O}\{\phi_1, \phi_2\} = \mathbf{O}[\phi_1 \wedge \phi_2] \neq \mathbf{O}\phi_1 \wedge \mathbf{O}\phi_2$.

For any objective formula ϕ, we call formulas $\mathbf{B}\phi$, not(ϕ), $\mathbf{O}\phi$ and their negations *belief literals*. Here $\mathbf{B}\phi$ and $\neg\mathbf{B}\phi$ are called **B**-literals, not(ϕ) and $\neg$not(ϕ) are called not-literals, $\mathbf{O}\phi$ and $\neg\mathbf{O}\phi$ are called **O**-literals. We call an MBNF theory Δ *simple* if Δ is a finite set of clauses each of which consists of a disjunction of belief literals and each is of the form

$$C: \neg\mathbf{B}(\phi) \vee \text{not}(\psi) \vee \mathbf{B}\phi_1 \vee \mathbf{B}\phi_2 \vee \ldots \vee \mathbf{B}\phi_n \vee \neg\text{not}(\psi_1) \vee \neg\text{not}(\psi_2) \vee \ldots \vee \neg\text{not}(\psi_m).$$

Δ is said to be *X-simple* if Δ is simple and for each clause $C \in \Delta$, the objective formulas ϕ and ϕ_i are conjunctions of propositional literals. Δ is said to be **B**-simple if Δ is simple and for each clause $C \in \Delta$, C does not contain the belief literal of the form $\neg\mathbf{B}\phi$. Note that simple theories are all subjective. The X-simple and B-simple classes include the MBNF theories that correspond to various forms of logic programs discussed in [8, 9]. the theories that correspond to circumscription, and the theories that

correspond to a large class of default theories, etc. The following translation embeds classes of X-simple and B-simple theories into OL.

Definition [1]. (mapping between MBNF and OL). Define the mapping π, which maps simple theories in MBNF to OL as follows. Let Δ be a simple theory in MBNF. The mapping π maps a belief literal in MBNF to a formula in OL:

$$\begin{array}{lll} \pi: & \mathbf{B}\phi & \rightarrow \phi \wedge \mathbf{B}\phi \\ & \neg\mathbf{B}\phi & \rightarrow \neg(\phi \wedge \mathbf{B}\phi) \\ & \text{not}(\phi) & \rightarrow \neg\mathbf{B}\phi \\ & \neg\text{not}(\phi) & \rightarrow \mathbf{B}\phi \end{array}$$

For a given clause $C \in \Delta$, $C = L_1 \vee L_2 \vee ... \vee L_k$, the mapping π defines the corresponding formula C′ in OL, which is of the form $C' = L_1' \vee ... \vee L_k'$ where each L_j' is obtained from L_j by applying the mapping π. Let the OL theory $\Delta' = \{C': C \in \Delta\}$. We call Δ' the *image* of Δ under π.

Note that in the above definition for π, the belief literal $\mathbf{B}\phi$ on the left hand side of the "$\rightarrow$" is in MBNF and the formula $\phi \wedge \mathbf{B}\phi$ on the right hand side is an OL formula. Assume $\Delta = \{\neg\text{not(p)} \vee \mathbf{B}q, \text{not(r)} \vee \neg\mathbf{B}s\}$. Then the image of Δ is $\Delta' = \{\mathbf{B}p \vee (q \wedge \mathbf{B}q), \neg\mathbf{B}r \vee \neg\mathbf{B}s \vee \neg s\}$.

Theorem 1. [1]. Let Δ be an X-simple or B-simple MBNF theory and let Δ' be its image in the OL logic. Let Δ_{OL} be $\mathbf{O}\Delta'$ if $\langle I, \emptyset\rangle$ are models of Δ otherwise let Δ_{OL} be $\mathbf{O}\Delta' \wedge \neg\mathbf{B}\Box$. Then a structure $\langle I, S\rangle$ is an MBNF model of Δ if and only if it is an OL model of Δ_{OL}.

According to Theorem 1, the classes of X-simple and B-simple theories can be embeded in OL. In [1], we showed that the detection of whether structures of the form $\langle I, \emptyset\rangle$ are models of Δ is decidable, using the resolution method in [5]. Details are omitted here.

4. Between OL and Other Logical Formalisms

Once the relationship between OL and MBNF is clarified, we can proceed to clarify the connection of OL with various non-monotonic logics as well as logic programming, since MBNF is known [8, 9] to be closely related to those formalisms.

4.1 Relations to AE logic, Default Logic and Circumscription

The AE logic was originally introduced by Moore [15]. An AE logic theory Δ is nothing but a *basic* theory in OL. (Here we substitute the modal operator "**B**" in OL for the operator "**L**" used in the original definition of AE logic). Such a theory intends to capture an agent's self-knowledge or beliefs. The notion of *stable expansion* plays a central role in AE logic. Intuitively, a stable expansion T of an AE

theory A is a meaningful set of beliefs which can be held by a rational agent, who has A as initial assumptions. Given an AE theory A and an AE formula ϕ, the inferencing problem in AE logic is whether or not ϕ is in every stable expansion of A. The link between the belief set of a model of $O\Delta$ and a stable expansion of Δ (as an AE logic theory) was established by Levesque. Let $W \subseteq \Omega$ be a set of worlds. The *belief set* for W is defined to be Belief(W) = $\{\phi \mid B\phi$ is true in OL logic in the structure <w, W> for any $w \in \Omega\}$. The following theorem shows that the inferencing problem in AE logic is straightforwardly done in OL logic - ϕ is in every stable expansion of Δ if and only if $O\Delta \vdash B\phi$.

Theorem [6]. Let Δ be an AE logic theory. A set W of worlds is a model of $O\Delta$ if and only if Belief(W) is a stable expansion of Δ.

Default logic defined by Reiter [17] is another important non-monotonic logic. A default logic theory Δ = <F, D> is a pair F and D, where F is a propositional theory and D is a set of defaults, each of the form $\frac{\alpha: \beta}{\gamma}$. The meaning of the default $\frac{\alpha: \beta}{\gamma}$ is informally the following: If α is provable and β is consistent with what is provable, then infer γ. The notion of *extension* in default logic plays the role similar to that of stable expansion in AE logic. A set of propositional formulas E is an extension of a default theory Δ if E is a fixed-point of some operator Γ.

Clearly, AE logic and default logic are closely related. Konolige made the first effort [4] relating the two together. He defines a translation which maps the default $\frac{\alpha: \beta}{\gamma}$ to the AE formula $B\alpha \wedge \neg B\neg\beta \rightarrow \gamma$. He showed that under this translation, each extension of the original default theory is the objective part of a stable expansion of the associated AE theory, but the converse may not be true. Marek and Truszczynski [12] improved on Konolige's results and gave a clear characterization between AE logic and default logic. They defined weak extensions of default logic, strong and robust expansions of AE logic, and showed that weak extensions correspond to precisely stable expansions and robust expansions correspond to extensions. Here we take an approach different from both [4] and [12]. We characterize a subclass of default theories and define a slightly different translation such that extensions correspond to exactly stable expansions for the subclass of default theories, under the new translation.

Definition. Let Δ = <F, D> be a default theory. Define the OL translation of Δ to be the theory $OL(\Delta) = F \cup D'$, where $D' = \{\alpha \wedge B\alpha \wedge \neg B\neg\beta \rightarrow \gamma \mid \frac{\alpha: \beta}{\gamma} \in D\}$.

Theorem 2. Let Δ = <F, D> be a default theory such that F is consistent and Δ satisfies one of the two conditions:

(1) Each default in D has no prerequisite, i.e., each default is of the form $\frac{:\beta}{\gamma}$.

(2) F is a conjunction of literals, and for each default $\frac{\alpha:\beta}{\gamma} \in$ D, α and γ are conjunctions of literals.

Let OL(Δ) be the OL translation of Δ. Then any set of objective formulas E is an extension of Δ if and only if E is the objective part of a consistent stable expansion of OL(Δ). Consequently, an objective formula ϕ is in every extension of Δ if and only if $\mathbf{B}\phi$ is true in every model of O[OL(Δ)].

Now we turn to the links between OL and circumscription. Circumscription defined by McCarthy [14] is a formalism for non-monotonic reasoning which is based on the notion of minimal models. Let A(P) be a (consistent) first order theory with predicate symbol P occurring in A(P). The circumscription of P in A(P), denoted as CIRC(A(P), P) is defined to be the theory whose models are precisely the models of A(P) in which the extension of P is *minimized*. Lifschitz has shown [8, 9] that for a (first order or propositional) theory A(P), the circumscription of P in A(P), CIRC(A(P), P), can be captured by the MBNF models of the theory $\mathbf{BA(P)} \wedge (\mathbf{not(P(x))} \rightarrow \mathbf{B}\neg\mathbf{P(x)})$. Correspondingly, we have the following result:

Theorem 3. Let A(P) be a propositional theory. For any objective formula ϕ, ϕ is true in all models of CIRC(A(P), P) if and only if $\mathbf{B}\phi$ is true in all models of the theory $\mathbf{O}[\mathbf{A(P)} \wedge \mathbf{BA(P)} \wedge (\neg\mathbf{BP} \rightarrow \neg\mathbf{P} \wedge \mathbf{B}\neg\mathbf{P})]$.

4.2 Relation to Logic Programming

The relationship between various non-monotonic logics and logic programming has been studied by a number of researchers [2, 3, 16, 18]. The AE logic has been found to have close connection with the *stable model* semantics of normal programs [2]. It has been found that default logic can also provide a natural interpretation of stable model semantics. However, for the extended logic programs (say, with classical negations as defined by Gelfond and Lifschitz [3]), no existing approach gives a natural interpretation of such programs as AE theories, while there exists interpretation of the extended programs into default theories. Therefore it may appear that default logic (not the AE logic), is *the* one that "correctly" extends the stable semantics in logic programming to handle classical negation. Our result in this subsection shows, however, AE logic *can* capture the semantics of extended logic programming. This conclusion concurs with the most recent results of other researchers [11, 13].

For the sake of completeness, we also state in this subsection the (existing) result relating OL logic and stable model semantics. Recall a normal logic program is a finite set of rules of the form " a $\leftarrow b_1, b_2, ..., b_k$, not c_1, not c_2, ..., not c_n.", where a, and each b_j, c_k are all atoms and "not" denotes *negation as failure*. Define the OL theory I(P)to be the set of formulas obtained by converting each above rule into the

formula "a $\leftarrow b_1 \wedge b_2 \wedge ... \wedge b_k \wedge \neg \mathbf{B}c_1 \wedge ... \wedge \neg \mathbf{B}c_n$". For a given program P, we use HB(P) to denote the Herbrand Base of P, i.e., the set of all atoms appearing in P.

Theorem 4. Let P be a normal logic program and let I(P) be as defined above. Then a set of atoms M is a stable model of P, if and only if M = Belief(W) $\cap$ HB(P) for a model W of O[I(P)], if and only if M = E $\cap$ HB(P), where E is a stable expansion of I(P).

Gelfond and Lifschitz [3] proposed the extended logic programs to include classical negation in addition to negation as failure in logic programming. An extended program P is a set of extended rules each is of the form

$$r: l_1 \mid l_2 \mid ... \mid l_r \leftarrow l_{r+1}, l_{r+2}, ..., l_k, \text{not } l_{k+1}, \text{not } l_{k+2}, ..., \text{not } l_n.$$

Here each l_j is a propositional literal and "not" denotes negation as failure. For such extended programs, the notion of an *answer-set* is defined which forms the semantical foundation for question-answering. An answer-set of a program P is a set of propositional literals defined by a fixed-point operation [3]. Gelfond and Lifschitz showed that the extended disjunctive program can be embeded in default logic. They also indicated that embedding such programs into AE logic seemed to be not so straightforward. In the following, we give a simple, syntactical translation which embeds extended disjunctive logic program to OL logic (and hence to AE logic).

Let P be disjunctive logic program with classical negation. Translate the above extended rule r into the OL formula "$l_1 \wedge \mathbf{B}l_1 \vee l_2 \wedge \mathbf{B}l_2 \vee ... \vee l_r \wedge \mathbf{B}l_r \leftarrow l_{r+1} \wedge \mathbf{B}l_{r+1} \wedge l_{r+2} \wedge \mathbf{B}l_{r+2} \wedge ... \wedge l_k \wedge \mathbf{B}l_k \wedge \neg \mathbf{B}l_{k+1} \wedge ... \wedge \neg \mathbf{B}l_n$". Define the OL translation of P to be the theory OL(P) which consists of the all the OL formulas obtained by such translation.

Theorem 5. Let P be a disjunctive logic program with classical negation and let OL(P) be the associated OL theory. Then all models of O[OL(P)] are precisely structures of the form <w, Mod(ANS)>, where ANS is an answer-set of P.

Acknowledgement

I am grateful to Vladimir Lifschitz and Grigori Schwarz, Wiktor Marek and Miroslaw Truszczynski for sending me their recent papers.

References

[1] J. Chen, Minimal Knowledge + Negation as Failure = Only Knowing (sometimes), To appear in *Proceedings of 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning*, 1993.

[2] M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming, *Proceedings of the 5th International Conference on Logic Programming*,

1988, pp. 1070-1080.

[3] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, **9**, 1991, pp. 365-385.

[4] K. Konolige, On the Relation between Default and Autoepistemic Logic, *Artificial Intelligence*, **35** (3), 1988, pp. 343-382.

[5] S. Kundu, A New Logic of Beliefs: Monotonic and Non-monotonic Beliefs - Part I, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991, pp. 486-491.

[6] H. J. Levesque, All I Know: A Study in Autoepistemic Logic, *Artificial Intelligence*, **42** (1-2), 1990, pp. 263-309.

[7] H. J. Levesque, Foundations of a Functional Approach to Knowledge Representation, *Artificial Intelligence*, **23** (2), 1984, pp. 155-212.

[8] V. Lifschitz, Nonmonotonic Databases and Epistemic Queries, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991, pp. 381-386.

[9] V. Lifschitz, Minimal Belief and Negation As Failure, Submitted for publication, 1992.

[10] V. Lifschitz, Between Circumscription and Autoepistemic Logic, *Proceedings of the 1st International Conference on Knowledge Representation and Reasoning*, 1989, pp. 235-244.

[11] V. Lifschitz, G. Schwarz, Extended Logic Programs and Autoepistemic Theories, To appear in *Proceedings of 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning*, 1993.

[12] W. Marek, M. Truszczynski, Relating Autoepistemic and Default Logics, *Proceedings of the 1st International Conference on Knowledge Representation and Reasoning*, 1989, pp. 276-288.

[13] W. Marek, M. Truszczynski, Reflexive Autoepistemic Logic and Logic Programming, To appear in *Proceedings of 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning*, 1993.

[14] J. McCarthy, Applications of Circumscription to Formalizing Common Sense Knowledge, *Artificial Intelligence*, **26** (3), 1986, pp. 89-116.

[15] R. Moore, Semantical Considerations on Nonmonotonic Logic, *Artificial Intelligence*, **25** (1), 1985, pp. 75-94.

[16] T. Przymusinski, On the relationship Between Logic Programming and Nonmonotonic Reasoning, *Proceedings of AAAI-88*, 1988, pp. 444-448.

[17] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, **13** (1-2), 1980, pp. 81-132.

[18] W. Truszczynski, Embedding Default Logic into Modal Non-monotonic Logics, Proceedings of the 1st International Workshop on Logic Programming and Non-monotonic Reasoning, 1991, pp. 151-165.

Terminological Logic Involving Time and Evolution: A Preliminary Report *

Patrick Lambrix and Ralph Rönnquist

Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden
patla@ida.liu.se ralro@ida.liu.se +46 13 281979

Abstract. Although terminological logics as well as temporal reasoning has received considerable attention in the knowledge representation community in the last two years, few attempts have been made to integrate these fields. We study the combination of the temporal logic LITE and a terminological logic to obtain a temporal terminological logic. We emphasize defining a terminological logic (T-LITE) where the extensions of concepts are time-dependent in the following sense: first, the individuals belonging to a concept are appearances of objects in a temporal context; secondly, we allow concepts to be defined in terms of developments of objects. A formal semantics for T-LITE is provided.

1 Introduction

Although terminological logics as well as temporal reasoning has received considerable attention in the knowledge representation community in the last years, few attempts have been made to integrate those fields. We study the combination of LITE (Logic Involving Time and Evolution) semantics and a base terminological logic so as to extend terminological logic with concepts regarding time and evolution.

Terminological logics or concept languages are languages tailored for expressing knowledge about concepts and concept hierarchies. They are usually given a Tarski style declarative semantics (see eg. [Neb90, App A]), which allows them to be seen as sub-languages of predicate logic. One starts with primitive concepts and roles, and can use the language constructs (such as intersection, union, role quantification etc.) to define new concepts and roles. Concepts can be considered as unary predicates which are interpreted as sets of individuals whereas roles are binary predicates which are interpreted as binary relations between individuals. The basic reasoning tasks are unsatisfiability and subsumption checking. A concept is unsatisfiable if it always denotes an empty set. A concept C is subsumed by a concept D if the extension of C is always a subset of the extension of D.

* This work is supported by funds from the Swedish Institute for Technical Development, under grant # 8802819P

A whole family of knowledge representation systems have been built using these languages and for most of them complexity results for the subsumption algorithm are known (e.g. KL-ONE[3], BACK [8], CLASSIC [2], KANDOR [9], KRYPTON [4], LOOM [7]).

'LITE' ([11],[12]) on the other hand, is a variation to first-order predicate logic where in particular the notion of *object* is revised from being an indivisible entity into being a temporal structure of *versions*. Each object version is then indivisible and unchanged in time, while an object changes in time by taking different appearances.

When approaching the task of mixing terminological logic with time, we observe that the idea of "temporal concept" requires some analysis. On the one hand, a concept may be time-dependent in its extension, i.e. the set of objects satisfying its definition is different at different times. This is the case usually dealt with (e.g. [13]), and it seems to be in some way a "first" or more immediate case.

A concept may also be time-dependent in its definition, whereby we mean that an object is included in or excluded from the concept extension depending on which development the object shows. For instance, the concept 'traffic-light' defined as a light cycling over being[2] green, yellow, and red is such a concept.

The latter kind of concepts seems to potentially extend beyond the idea of concept languages, because the extension of a concept no longer is a set of single objects but more like a set of history fragments involving several objects in some certain development combinations. For instance, the concept of 'chess-game' denotes scenarios involving (at least) two players, a chess board, and chess pieces, where the latter move over the chess board admitting to a small set of specific rules. To cover these kinds of concepts in terms of terminological logic, one has to reify; to introduce abstract objects that denote scenarios.

Finally, a concept may also change in itself, so that its extension changes because its definition changes. For instance the concept of (legally) 'adult' may change by decreasing the required age. This kind of development is probably even more difficult to deal with within terminological logic, since for one thing, it means that subsumption relationships change.

In our study here we use the same base concept language as [13], which formally is a subset of most current terminological logics. However while the focus in [13] is on the ability to express extensional concept changes by using explicit time references, we focus on using more implicit temporal qualifications for object selection in the sense of classification[3]. Further we make a start in tackling the second kind of time-dependent concepts, i.e. where an object is included in or excluded from the concept extension depending on which development the object shows. In section 2 we describe the temporal logic LITE. Then we use the LITE semantics to define in section 3 the temporal terminological logic T-LITE. We conclude the paper with an example (section 4) and a discussion (section 5).

[2] Subject to national variation.

[3] It may of course be useful in practice to allow clock metrics, but in our approach, such metric is contained within the data rather than extending the representation language.

2 LITE

The LITE ([12]) logic is a first-order predicate logic with an extension allowing objects to be seen as sets of versions. The syntax of the temporal framework is the normal syntax of first-order predicate logic extended by the use of the temporal operator @. In principle, the normal first order logic semantics is also used, but over versions, rather than over objects.

Formally, a structure is a tuple (Vs,Obj,Col,Rel,Func). Vs is the set of distinct individuals that correspond to the appearances or versions of the objects. We have a partial time order between versions. Versions belonging to the same object which are not ordered by the partial time order are said to be parallel versions. Obj is the set of objects. An object is a set of versions. All objects are pairwise disjoint. Col is the set of collections. A collection is a set composed of exactly one version of each object such that all those versions are potentially contemporary (see below). The version of object x in the collection t is denoted by x@t. A collection can be seen as a possible time-slice over the objects. Func is the set of functions. A function has its arguments in Vs or Col and its image in Obj. Rel is the set of relations. A relation has its arguments in Vs or Col.

A formula is interpreted with respect to a structure S, a collection t in the structure and a binding h which maps variables to objects and time variables to collections. The principle is to interpret formulas in the classical way with the exception that object references are disambiguated to versions by intersecting the object with the collection t.

By revising the notion of object from being an indivisible entity into being a temporal structure of versions, mixed time relations are easily expressed in LITE. A sentence like "Now, I like young Plato but I don't like old Plato" can be expressed in LITE as follows.

$$\forall t : ((\text{Young}(\text{Plato@}t) \rightarrow \text{Like}(\text{me@now}, \text{Plato@}t)) \wedge (\text{Old}(\text{Plato@}t) \rightarrow \neg\text{Like}(\text{me@now}, \text{Plato@}t)))$$

We see that in this sentence three different tenses are at work, indicated by 'now', 'young' and 'old'. With temporal indexing on objects, 'now' qualifies 'I' while 'young' and 'old' are different qualifications for Plato.

Due to the importance of the 'collection' notion, we give the formal definition [12].

Definition 1. A set u of one version of each object is a *collection* iff there is no object x whose version $x@u$ in u is succeeded by a version α of x that precedes the version $y@u$ of another object y in u.

The time ordering of versions induces an ordering on the collections such that $t_1 \preceq t_2$ for collections t_1 and t_2 if for all objects x: $x@t_1 \preceq x@t_2$. We note that not all collections are related to each other in this way. If $t_1 \not\preceq t_2$ and $t_2 \not\preceq t_1$ then t_1 and t_2 are regarded as parallel collections. This could for instance represent different possible ways in which the world can have evolved. Let us assume for the example in figure 1 that $t_1 = \{x1, y1\}$, $t_2 = \{x2, y2\}$ and $t_3 = \{x3, y2\}$.

Then we have $t_1 \preceq t_2$, $t_1 \preceq t_3$, and t_2 and t_3 are parallel collections. An interval $[t_1, t_2]$ for $t_1 \preceq t_2$ is the set of collections t such that $t_1 \preceq t \preceq t_2$.

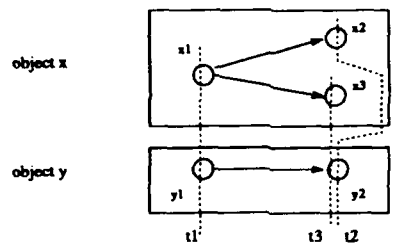


Fig. 1. Ordering between collections

In the rest of this paper we sometimes write "an object at time t" where we actually mean "the version of the object in collection t".

3 A Temporal Terminological Logic

In our attempt to combine the LITE semantics with terminological logic, we define a term valuation function relative to a structure so that the value of a concept is a set of pairs $\langle d, t \rangle$ of objects d and collections t. We call such a pair a *pointer* which points to the version $d@t$.

Concept Terms

concept ::= *primitive − concept*
| (**and** *concept*$^+$)
| (**all** *role concept*)
| (**atleast** *min role*)
| (**atmost** *max role*)
| (**sometimes** *concept*)
| (**always** *concept*)
| (**at** *concept*)
| (**when** *role concept*)
| (**does** *role*)

Role Terms

role ::= *primitive − role*
| (**and** *role*$^+$)
| (**domain** *concept*)
| (**range** *concept*)
| (**occurs** *role*)
| (**prevails** *role*)
| (**seq** *role*$^+$)
| (**loop** *role*$^+$)

Fig. 2. Definitions of concept and role terms.

A concept term is a term whose value is a set of pointers; namely the pointers satisfying the particular selection conditions of the term. A role term is a term whose value is a set of pointer pairs such that the pairs satisfy the role term definition. Our language accepts the concept and role terms shown in figure 2. Their formal semantics are defined as in figures 3 and 4.

Standard Terms

D is the domain set of all pointers.

$\epsilon[(\textbf{and } c_1 \ \cdots \ c_n)] = \bigcap_{i=1}^{n} \epsilon[c_i]$

$\epsilon[(\textbf{all } r \ c)] = \{(d,t) \mid \forall d', t' : \langle (d,t),(d',t') \rangle \in \epsilon[r] \rightarrow (d',t') \in \epsilon[c]\}$

$\epsilon[(\textbf{atleast } m \ r)] = \{(d,t) \mid \sharp I(d,t,r) \geq m\}$ where $I(d,t,r) = \{d' \mid \langle (d,t),(d',t) \rangle \in \epsilon[r]\}$

$\epsilon[(\textbf{atmost } m \ r)] = \{(d,t) \mid \sharp I(d,t,r) \leq m\}$ where $I(d,t,r) = \{d' \mid \langle (d,t),(d',t) \rangle \in \epsilon[r]\}$

$\epsilon[(\textbf{and } r_1 \ \cdots \ r_n)] = \bigcap_{i=1}^{n} \epsilon[r_i]$

$\epsilon[(\textbf{domain } c)] = \epsilon[c] \times D$

$\epsilon[(\textbf{range } c)] = D \times \epsilon[c]$

Primary Temporal Terms

$\epsilon[(\textbf{sometimes } c)] = \{(d,t) \mid \exists t' : (d,t') \in \epsilon[c]\}$

$\epsilon[(\textbf{always } c)] = \{(d,t) \mid \forall t' : (d,t') \in \epsilon[c]\}$

$\epsilon[(\textbf{at } c)] = \{(d,t) \mid \exists d' : (d',t) \in \epsilon[c]\}$

$\epsilon[(\textbf{when } r \ c)] = \{(d,t) \mid \forall d' : \langle (d,t),(d',t) \rangle \in \epsilon[r] \rightarrow (d',t) \in \epsilon[c]\}$

$\epsilon[(\textbf{occurs } r)] = \{\langle (d,t),(d',t) \rangle \mid \exists t' : d@t = d@t' \wedge \langle (d,t'),(d',t') \rangle \in \epsilon[r]\}$

$\epsilon[(\textbf{prevails } r)] = \{\langle (d,t),(d',t) \rangle \mid \forall t' : d@t = d@t' \rightarrow \langle (d,t'),(d',t') \rangle \in \epsilon[r]\}$

Fig. 3. Standard Terms and Primary Temporal Terms.

Secondary Temporal Terms

$\epsilon[(\textbf{seq } r_1 \ \cdots \ r_n)] = \{\langle (d,t),(d,t') \rangle \mid \exists t_1 \preceq \cdots \preceq t_{n+1} : t = t_1 \wedge t' = t_{n+1} \wedge (\forall i \in \{1,\ldots,n\} : \langle (d,t_i),(d,t_{i+1}) \rangle \in \epsilon[r_i])\}$

$\epsilon[(\textbf{loop } r_1 \ \cdots \ r_n)] = \{\langle (d,t),(d,t') \rangle \mid \exists k > 0, t_1 \preceq \cdots \preceq t_{(k*n+1)} : t = t_1 \wedge t' = t_{(k*n+1)} \wedge (\forall i \in \{1,\ldots,k*n\} : \langle (d,t_i),(d,t_{i+1}) \rangle \in \epsilon[r_{q(i)}])\}$

where $q(i) = (i + p \bmod n) + 1$ for some constant $p \in \{0,\ldots,n-1\}$

$\epsilon[(\textbf{does } r)] = \{(d,t) \mid \exists t', t'' : t' \preceq t \preceq t'' \wedge \langle (d,t'),(d,t'') \rangle \in \epsilon[r]\}$

Fig. 4. Secondary Temporal Terms.

Concepts and roles may also be primitive[4], in which case their values are directly retrievable in the structure; the concepts as sets of pointers and the roles as sets of pairs of pointers. Intuitively a primitive concept is seen as a property of an object version, and a primitive role is seen as a binary relation between object versions.

The constructs **and**, **all**, **atleast**, and **atmost** for concepts and **and**, **domain**, and **range** for roles are regarded as standard terms and we aim for definitions which as much as possible follow the standard definitions for these terms. We do introduce however some amount of temporal dependency; e.g. the counting of role fillers implied by **atmost** and **atleast** occurs within collections.

Although these standard terms do not seem to be time dependent in the mnemonics of the operators, they really are of course. Consider for example the following term.

(**and** male (**atleast** 1 (**and** child (**range** female))))

This term selects the males in versions at the times at which they have atleast one daughter. By associating a concept with a set of pointers, we allow a concept to select versions of objects within a temporal context.

[4] In [8] a primitive concept is a concept for which the meaning of the concept can only be partially determined by its description. It seems to serve the same purpose in the sense that all concepts and roles are defined directly or indirectly from the primitive concepts and roles.

The constructs **sometimes**, **always**, **at**, and **when** for concepts and **occurs** and **prevails** for roles are seen as primary temporal terms. They are used for temporal qualifications based on inter object synchronization.

The constructs (**sometimes** c) and (**always** c) are object selectors, i.e. they select all pointers for a particular object if the object belongs to the concept c at some time, or at all times respectively.

The (**at** c) construct is the temporal[5] counterpart of (**sometimes** c). It selects a whole collection whenever some object in that collection satisfies c. The **at** term is for example most us 'ul in connection with more explicit collection references, so that e.g. *August-1990* would be a qualification for some collections.

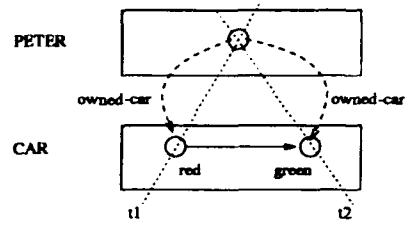


Fig. 5. Peter repaints his car.

The concept term (**when** r c) is in principle the same as (**all** r c) but with the addition of the constraint that the role filling pointer is synchronized with the collection in which the role is filled. The difference can easily be seen in the following example. Consider an object with the name PETER. Assume then that a particular version of PETER owns a red car that changes to green[6] and he owns no other cars (see figure 5). Then the following terms

(**all** owned-car red)
(**all** owned-car green)

do not select this particular PETER version, whereas the following terms

(**when** owned-car red)
(**when** owned-car green)

select both this particular PETER version although within different temporal context ($\langle$PETER, $t_1\rangle$ for (**when** owned-car red) and $\langle$PETER, $t_2\rangle$ for (**when** owned-car green)).

Whereas **sometimes** and **always** are object selectors with respect to concepts, **occurs** and **prevails** are version selectors with respect to roles. They

[5] In the sense that it selects collections instead of object.

[6] This means that we assume that PETER's appearance does not change when he sees his car changing color.

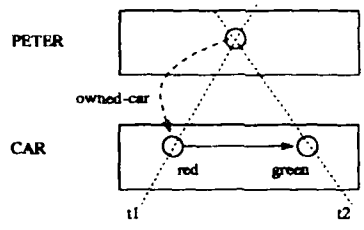


Fig. 6. Peter repaints and sells his car.

select all pointers for a particular version as first argument if the version is first argument for the role r at some time, or at all times respectively. The role term **occurs** extends a contemporary role filling to a contemporary role filling over a whole version. The role term **prevails** selects pairs only if the role holds for the whole version or the pointer which is first argument.
For the example in figure 5 both of the following terms

(**prevails** owned-car)
(**occurs** owned-car)

select $\langle\langle \text{PETER}, t_1\rangle, \langle \text{CAR}, t_1\rangle\rangle$ and $\langle\langle \text{PETER}, t_2\rangle, \langle \text{CAR}, t_2\rangle\rangle$.
For the example in figure 6, (**prevails** owned-car) does not select any pointer pair. On the other hand (**occurs** owned-car) extends the role to the whole version and selects $\langle\langle \text{PETER}, t_1\rangle, \langle \text{CAR}, t_1\rangle\rangle$ as well as $\langle\langle \text{PETER}, t_2\rangle, \langle \text{CAR}, t_2\rangle\rangle$.

Finally the constructs **does** for concepts and **seq** and **loop** for roles are introduced as secondary temporal terms allowing for temporal qualification based on intra object synchronization, i.e. object development. These operators might also be called plan forming operators.

The concept term **does** transforms a succession role into a selection of all the pointers over which the development appears. The outcome of a **seq** term is the transition from the first to the last object versions between which the indicated development appears. An object changing color from green to yellow to red in succession could for instance be defined by the following term:

(**seq** (**domain** green) (**domain** yellow) (**domain** red))

Informally, the **loop** construct recognizes a recurrent development corresponding to a succession of equal sequences.

4 Example

By this example we illustrate two specific issues in using TELLITE. One thing is the ability to recognize and classify objects by means of their development in time. The other thing is the ability to recognize and classify a compound scenario by

means of how the developments of components are synchronized. Both of these are issues that emphasize the temporal aspects in a scenario classification.

An intuitively simple kind of object recognized and classified through its development is a traffic light. Informally, a traffic light is a light which continuously changes color from green to yellow to red to green. This can be formulated in T-LITE as follows. (We use ::= as a definition symbol.)

traffic-light ::= (**and** light (**always** (**does** (**loop** (**domain** green)
(**domain** yellow)
(**domain** red)))))

Traffic lights are posted at the entries to an intersection so that some few lanes are controlled by each light. The intersection as a whole can then be classified through the synchronized behavior of the lights. We call an intersection 'safe' if the lights are synchronized so that crossing lanes never show green at the same time.

We introduce a 'conflict point' as an object being related to two or more lanes, that each is controlled by a traffic light. The conflict point is 'resolved' when there is at most one lane whose signal is green at any one time. We obtain the following :

resolved-point::= (**and** conflict-point
(**atmost** 1 (**occurs** (**and** lane (**range** (**when** signal green))))))

The appearance of the **occurs** term forces the cycling of green signals to be represented in the development of the resolved point. Only one lane is allowed to have a green signal per resolved point object version. Without this **occurs** term resolved points whose development are unrelated with respect to the green light signaling, are allowed.

A safe intersection is then defined as follows:

safe-intersection ::= (**and** intersection (**all** crossing (**always** resolved-point)))

We note that (**always** resolved-point) stresses the fact that every version of the resolved point object is classified as a resolved-point. Without the **always** the intersection might have been safe only sometimes, which is not our intention.

5 Discussion

An important part of a terminological logic is a subsumption algorithm - ideally sound, complete and tractable, although this ideal must usually be compromised ([8]). However, similarly to Schmiedel ([13]), we do at this point not have a subsumption algorithm. In this report, we have concentrated on extending the syntax and semantics of a basic terminological logic to include temporal information about changing extensions of an object and concepts defined in terms of development of objects. Subsumption algorithms for T-LITE can essentially use

the algorithms for our non-temporal base language for the standard terms. This is clear as except for the extra synchronization in **atleast** and **atmost**, we have a one to one mapping between the terms in the base language and the standard terms by using pointers as individuals. For the temporal terms we observe that our semantics give the intended subsumption relations. For instance, (**always** *c*) is subsumed by (**sometimes** *c*) and *c*. Also (**all** *r c*) is subsumed by (**when** *r c*) and (**seq** *r*1 ... *rn*) is subsumed by (**loop** *r*1 ... *rn*). Also similarly to Schmiedel ([13]), we do not expect complete and tractable subsumption algorithms. We may be content however with a tractable but incomplete algorithm, or with a complete algorithm with good average behavior. Further, [10] for instance, shows that a subsumption algorithm which is not complete with respect to standard semantics, may well be complete with respect to a weakened semantics.

Schmiedel's work [13] is based on Allen's interval calculus [1] as temporal framework. Concepts are associated with functions from a interval based time domain to sets of individuals. Roles are associated with functions from this interval based time domain to sets of pairs of individuals. (This means that the extension of a concept can change over time.) The use of Allen's interval calculus in defining Schmiedel's standard semantics, implies that if we know that a concept or relation holds over an interval we do not have automatically any information regarding whether the concept or role holds during any sub-interval. This problem can be solved by allowing to associate with a concept or role Shoham's hereditary properties [14]. In our approach we define concepts and roles using pointers, i.e. objects at a specific time where the specificity directly relates to object change. One advantage is that we have exploited LITE's ability to express mixed temporal relations. Therefore roles are allowed to relate objects in different temporal contexts. Further in our approach it is the structure itself which tells us whether we can conclude information about sub-intervals.

The secondary temporal terms have some resemblance with the use of the plan description forming operators of Devanbu and Litman [5]. Their system CLASP is used to describe and classify plans. Their 'plan description forming operators' include a SEQUENCE and LOOP operator. For instance

(SEQUENCE *a b c*)

represents a compound plan consisting of a sequence of a plan of type *a*, a plan of type *b* and a plan of type *c*. Our **seq** operator works on roles. A plan could be seen as a transition function between collections (and therefore a role where a pointer in one collection is associated with a pointer in the other collection involving the same object).

(**seq** *a b c*)

would then represent all the instantiations of the compound plan as a transition function.

Recurrence is a crucial element of our common-sense notion of time. As Koomen [6] expresses :

> From our earliest days we learn to perceive time as a result of two important cognitive abilities : the awareness of change in the world around us, and the ability to detect regularities in that change. Without change there is no awareness of anything, and without regularities there is awareness of chaos only.

However little work has been done on this topic. In [6] a first order axiomatization based on Allen's interval calculus [1] is proposed to reason about recurrence. By using the LITE semantics and our plan formation operators, we have made a first step in introducing reasoning about recurrence in terminological logics.

References

1. Allen, J.F., 'Maintaining Knowledge About Temporal Intervals', in *Communications of the ACM*, Vol 26(11), pp 832-843, 1983.
2. Borgida, A., Brachman, R.J., McGuinnes, D.L., Resnick, 'CLASSIC : A Structural Data Model for Objects', in *Proceedings of the International Conference on Management of Data - SIGMOD89*, pp 59-67, 1989.
3. Brachman, R.J., Schmolze, J.G., 'An Overview of the KL-ONE Knowledge Representation System', in *Cognitive Science*, 9(2), pp 171-216, 1985.
4. Brachman, R.J., Pigman Gilbert, V., Levesque, H.J., 'An Essential Hybrid Reasoning System : Knowledge and Symbol Level Accounts in KRYPTON', in *Proceedings of the International Joint Conference on Artificial Intelligence -IJCAI85*, pp 532-539, 1985.
5. Devanbu, P.T., Litman, D.J., 'Plan-Based Terminological Reasoning', in *Proceedings of the Conference on Representation of Knowledge - KR91*, pp 128-138, 1991.
6. Koomen, J.A.G.M., 'Reasoning about Recurrence', in *International Journal of Intelligent Systems*, Vol 6, pp 461-496, 1991.
7. MacGregor, R., Bates, R., 'The LOOM Representation Language', Technical Report ISI/RS-87-1988, University of Southern California, Information Science Institute, Marina del Rey, CA, 1987.
8. Nebel, B., *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence, 422, 1990.
9. Patel-Schneider, P., 'Small Can Be Beautiful in Knowledge Representation', in *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pp 11-16, 1984.
10. Patel-Schneider, P., 'Adding Number Restrictions to a Four-Valued Terminological Logic', in *Proceedings of the National Conference on Artificial Intelligence - AAAI88*, pp 485-490, 1988.
11. Rönnquist, R., 'A Logic for Propagation Based Characterisation of Process Behaviour', in *Proceedings of the International Symposium on Methodologies for Intelligent Systems - ISMIS90*, pp 297-304, 1990.
12. Rönnquist, R., *Theory and Practice of Tense-bound Object References*, Ph.D. thesis, nr 270, Dpt. of Computer Science, Linköping University, 1992.
13. Schmiedel, A., 'A Temporal Terminological Logic', in *Proceedings of the National Conference on Artificial Intelligence - AAAI90*, pp 640-645, 1990.
14. Shoham, Y., 'Temporal Logics in AI : Semantical and Ontological Considerations', in *Artificial Intelligence*, Vol 33(1), pp 89-104, 1987.

KNOWLEDGE MANAGEMENT BY EXAMPLE

LEVENT V. ORMAN

Cornell University, Malott Hall, Ithaca, NY 14853

email: orman@crnlgsm

Abstract. Knowledge has many components such as data, constraints, queries, transactions, and derivation rules. Data is the only component that can be managed effectively in large quantities. All other components are in their infancy in terms of tools and techniques for efficient storage and retrieval, implementation and execution, and user specification and design. One approach to manage all components of knowledge in large quantities is to reduce them all to data. Many components of knowledge can be expressed in terms of examples, and examples are data. As such, all these components can be stored and retrieved efficiently in large quantities, their execution reduces to data comparison and can be done in parallel, and they can be specified, designed, and modified by end users since examples are more intuitive and easy to manipulate than general procedures.

Keywords: Knowledge engineering, Knowledge base management, Knowledge representation, Database constraints, Integrity maintenance, Rule base.

1. Introduction

Knowledge has many components. Data, constraints, queries, rules and transactions all have to be specified and designed, stored and organized for efficient retrieval, and implemented and executed when necessary. Data is the simplest and most thoroughly studied component of knowledge. In large quantities, storage and retrieval of all components of knowledge are problematic except data. Rule based systems for example, have no efficient way of organizing a large number of rules on secondary storage devices, and many require all rules to be in the main memory. Moreover, searches through a large number of rules to find matches or inconsistencies are often sequential since no efficient indexing or modeling capabilities exist [10, 12].

Data is the only component of knowledge,that can be designed by end users, using a simple data model consisting of flat files. All other components need to be specified and designed as procedures or as logical statements, both of which require professionally trained personnel such as analysts or knowledge engineers. Data is the only component of knowledge that can be organized and structured for efficient retrieval from secondary storage devices. Moreover, all such structuring has been automated by database management systems, relieving the users from the task of designing and optimizing each individual system separately. Finally, execution of requests for data has been studied extensively, with a heavy emphasis on optimization of such requests, while such optimization is in its infancy for all other components [2, 4, 13].

Since data is the only component of knowledge that is well understood and successfully studied, representing all components of knowledge as data has significant and immediate potential in alleviating all the aforementioned problems. Most components of knowledge can be represented in terms of examples; and examples are data. As such they can be stored and structured for efficient retrieval as data. Their execution often reduces to data comparison, which is simpler than procedural execution, and much simpler than general inferencing with rules. Lastly, examples are much more intuitive for end users to understand and specify correctly than many end user languages. In the following sections major components of knowledge will be expressed as example data, and studied as data [11, 16].

2. Constraints

Database constraints are logical statements that have to be obeyed by the data at all times [1, 2, 8, 15]. We restrict ourselves to first order logic, and a closed world database where all facts not in the database are assumed false. Constraints are critical in policing the database and weeding out incorrect and inconsistent data. They are usually implemented as procedures to be executed against the database, and after each modification to the database. However, they slow down the database maintenance process considerably, and in realistic large numbers they can bring the system operations to a complete halt. Consequently, their execution is often delayed and done in batches, leading to long periods of time when the correctness and the consistency of the database cannot be guaranteed. The inefficiency of database constraints emanate from three sources:

a. The inherent difficulty of executing these complex procedures against large databases [15].

b. The need to check all constraints for every modification to the database, since there is no efficient method to structure and organize the constraints so that only a small number of relevant constraints could be identified and retrieved for testing after each database modification. The attempts to ameliorate this problem produced minor improvements. One approach involves clustering constraints with respect to the files they reference, since a transaction involving a file F can only impact the constraints that reference F [10]. Another approach involves declarative specification of constraints, and theorem proving techniques to locate the constraints that may match a term of the transaction. This approach requires all constraints to be brought into the main memory and tested for matching terms, since there is no effective storage structure for constraints to accomplish the term matching directly on the secondary storage devices [6,14].

c. The difficulty of maintaining the constraints themselves as they change over time, since their semantics is buried within procedures, and often they can only be changed by completely rewriting them. This problem is partially overcome by declarative specification of constraints; however, both declarative specifications and procedures have to be stored as text, and

identifying and retrieving the constraints affected by a change in requirements is still a significant problem [5, 14].

Expressing constraints as example data attacks all three problems simultaneously. Examples are intuitive to end users and can be specified easily without extensive professional help; examples can be stored as data, structured into database files, and retrieved efficiently; and finally they can be enforced efficiently since enforcement reduces to data comparison between constraints as example data and the database.

Constraints can be expressed as example data that violate the integrity of the database. Given the following relational database for a university environment:

STUDENT (name, dept.)

COURSE (course#, dept.)

REG (name, course#)

containing the names and departments of students, course#, and departments of courses, and courses registered for by each student. Also given are the following database constraints:

C1: The students who take CS100 also take CS200.

C2: All CS students take all CS courses.

They can be expressed as examples violating the database as follows:

C1: A student x takes CS100 but not CS200.

REG(x, CS100), -REG(x, CS200)

C2: There is a CS student x who does not take a CS course y.

STUDENT(x, CS), COURSE(y, CS), -REG(x,y)

Clearly, variables such as x and y stand for unknown data items that would violate the integrity of the database if any data item existed to take their place.

Once the constraints are expressed as example data, they can be placed in a constraint base. The constraint base contains the same files and the same structure as the database except for a constraint# field attached to each file. Constraint# field contains a unique identifier for each constraint. The constraint base also contains negative files such as -REG; however, these negative files can be included in the positive files such as REG by merely attaching a negative sign to the constraint# of those negative records. The constraint base for the above constraints is shown below where the constraint# field is separated only for visual appeal:

constraint#	name	dept
2	x	CS

STUDENT

constraint#	course#	dept
2	y	CS

COURSE

constraint#	name	course#
1	x	CS100
-1	x	CS200
-2	x	y

REG

In this constraint base environment, locating and retrieving the constraints relevant to a particular database transaction is a simple database operation. One only needs to search for the constraints that match the data records involved. The only assumption needed for matching is that the variables (such as x) match all constants. In a file with m attributes and n records and assuming a binary search for each attribute, at most mlog2n records need to be checked to find a match. This number is much smaller than mn/2 checks on average, required by a sequential search, and can be improved even further by utilizing a multiattribute search technique.

Example: A transaction involving STUDENT(SMITH, CS) can only effect the constraint2 since it matches only the student record STUDENT(x, CS). A transaction involving REG(SMITH, CS100) may effect constraints 1 and 2, since it matches the REG records REG(x, CS100) and REG(x,y).

The execution of constraints involves comparing the constraint base to the database to find a match indicating a violation of the database integrity. The violations can be detected by creating a violation file V for each constraint where V contains a column for each variable in the constraint, a positive record for each match between the positive records of the constraint base and the database; and a negative record for each match between the negative records of the constraint base and the database. Subtracting negative records from the positive records leaves only the violations in the V file. Each record in the V file is a violation of the database integrity.

Example: Given the following university database:

SMITH	CS
JONES	CS

STUDENT

CS100	CS
CS200	CS

COURSE

SMITH	CS100
SMITH	CS200
JONES	CS100

REG

Constraint C1 is REG(x, CS100), -REG(x, CS200) The corresponding V1 is:

	x
1	SMITH
1	JONES
-1	SMITH

V1

x
JONES

V1

since REG(x, CS100) matches in the database REG(SMITH, CS100) and REG(JONES, CS100), and -REG(x, CS200) matches in the database the record REG(SMITH, CS200). Subtracting the negative records from the positive records, we get JONES. Intuitively, JONES is a violation of C1 since he takes CS100 but not CS200.

Similarly, constraint2 is STUDENT(x, CS), COURSE(y, CS), -REG(x, y). The corresponding V2 is:

	x	y
2	SMITH	CS100
2	SMITH	CS200
2	JONES	CS100
2	JONES	CS200
-2	SMITH	CS100
-2	SMITH	CS200
-2	JONES	CS100

V2

	x	y
2	JONES	CS200

V2

since STUDENT(x, CS) matches STUDENT(SMITH, CS) and STUDENT(JONES, CS), and COURSE(y, CS) matches the records COURSE(CS100, CS) and COURSE(CS200, CS), and -REG(x,y) matches all the records in REG. Subtracting the negative records from the positive ones we get JONES CS200. Intuitively

JONES CS200 is a violation of C2 since JONES is a CS student but fails to take a CS course, namely CS200.

In general, not all constraints need to be checked for every transaction. Integrity can be maintained incrementally, by assuming integrity before a transaction, and ensuring that the transaction does not violate it. Integrity maintenance then involves two separate tasks.

a. The identification and retrieval of relevant constraints for each transaction.

b. The execution of those constraints against the database to detect possible integrity violations. Moreover, the execution does not need to involve the whole database, and the identification and retrieval of the relevant portion of the database is another possible source of efficiency.

Both insertions and deletions of records into the database can be handled by merely matching the records involved against the constraint base, identifying the constraints involved. The only additional assumption is that the insertions are required to match only the positive records, and deletions match only the negative records. Once the relevant constraints are identified, a V file is created for each to detect violations. The database is also restricted only to the data records involved.

Example: Given the constraints:

C1: REG(x, CS100), -REG(x, CS200)

C2: STUDENT(x, CS), COURSE(y, CS), -REG(x,y)

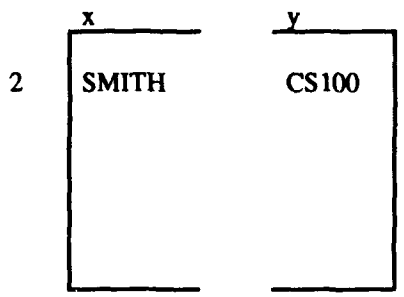
Insertion into REG file the record REG(DOE, CS200) fails to match any constraint records since the only records that match CS200 are negative ones. Consequently, no violations can occur. No further checks are necessary. Similarly, deleting from the STUDENT file the record (SMITH, CS) can cause no violations since -STUDENT(SMITH, CS) fails to match any constraint records. On the other hand, inserting into STUDENT the record STUDENT(DOE, CS) will cause a match with STUDENT(x, CS), and hence the constraint C2 may be violated. Restricting x=DOE, and forming V2:

	x	y
2	DOE	CS100
2	DOE	CS200

V2

which indicates a violation. Intuitively, the violation follows from the fact that all CS students have to take all CS courses. DOE is a CS student, but takes neither CS 100 nor CS200. Note that V2 is a restricted V file. It does not need to be fully developed for a given transaction, but it is restricted to the matches found, x=DOE in this case, resulting in a very efficient enforcement algorithm.

Similarly, deleting the record REG(SMITH, CS100) may violate the constraint C2, since -REG(SMITH, CS100) matches -REG(x, y). Restricting x=SMITH and y=CS100, and developing V2:



V2

V2 does not contain a record -2(SMITH, CS100) since it has been deleted, and hence a violation results. The violation follows intuitively from the fact that a CS student (SMITH) fails to take a CS course (CS100).

3. Extensions

In the general case, a constraint language requires the power of first order logic, and that power can be acquired by extending the constraint base with negative variables such as -x meaning "no such x" [7]. These constraints are less frequent in real life databases, however the constraint base can be extended to handle them for theoretical completeness.

Example: The following constraints indicate a violation of the database integrity if:

C3: There is a CS student taking no courses.

STUDENT(x, CS), REG(x, -y)

C4: Every CS student takes at least one course (i.e. let x be a student taking no CS courses; there is no such CS student).

STUDENT (-x, CS), REG(x, -y)

Such general constraints can easily be stored in a constraint base. The only additional requirement is the storage of negative variables. The constraint base containing C3 and C4 is shown below:

3	x	CS
4	-x	CS

STUDENT

3	x	-y
4	x	-y

REG

BIBLIOGRAPHY

1. *Goldstein*, R.C., *Storey* V.C., Unraveling IS-A Structures. Information Systems Research 3, 2, 1992.

2. Hammer, M., McLeod D., Database Description With SDM: A Semantic Data Model, Transactions on Database Systems, 6, 3, 351-386, 1981.

3. Hull, R., Yap C.K., The Format Model: A Theory of Database Organization, Journal of ACM, 31, 3, 518-537,1984.

4. Lochovsky, F.H., Tsichritzis, D.C., Data Models, Prentice Hall, 1982.

5. Morgenstem, M., Constraint Equations: Declarative Expression of Constraints with Automatic Enforcement, VLDB 33-42,1984.

6. Nicholas, J.M., Logic for Improving Integrity Checking in Relational Databases, Acta Informatica 18, 227-253,1982.

7. Orman, L.V., Functional Development of Database Applications, Transactions on Software Engineering 14, 9,1280-1292,1988.

8. Orman, L.V., Constraint Maintenance as a Database Design Criterion, Computer Journal 34, 1, 73-79, 1991.

9. Shephard, A., Kerschberg, L., PRISM: A Knowledge Based System for Semantic Integrity Specification and Enforcement in Database Systems, Proceedings of SIGMOD Conference 307-315, 1984.

10. Stonebraker, M., et al., The Postgres Rules System, Transactions on Software Engineering 14, 7, 1988.

11. Tansel, A.U., Arkun, M.E., Ozsoyoglu G., Time By Example Query Language for Historical Databases, Transactions on Software Engineering 15, 4, 464-478,1989.

12. Tsur, S., Zaniolo C., A Logic-Based Data Language, VLDB, 1986.

13. Ullman, J.D., Principles of Data and Knowledge Base Systems, Computer Science Press, 1989.

14. Urban, S.D., Delcambre, L.M.L., Constraint Analysis: Specifying Operations on Objects, Transactions on Knowledge and Data Engineering 2, 4, 391-400, 1990.

15. Weber, R., EDP Auditing, McGraw Hill, 1982.

16. Zloof, M.M., Query by Example: A Database Language, IBM Systems Journal 16, 4,1977.

System Reorganization and Load Balancing of Parallel Database Rule Processing *

Hasanat M. Dewan Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York, NY 10027, USA

Abstract

In the coming decade, high-speed network computing using processors that are orders of magnitude faster than the platforms available today, will enable the integration and coalescing of vast amounts of information stored in diverse databases. This will provide unprecedented new opportunities for *acquiring new knowledge* by applying various inferential processes against such massive databases. Meeting this challenge requires significant advances in our understanding of how to build efficient, high-performance knowledge-base systems targeted to run on a variety of parallel and distributed hardware architectures.

We address these concerns in the context of the PARADISER distributed rule processing system. We present an approach that combines statically computed *restrictions* on rule programs to partition the workload of rule evaluation among an arbitrary number of processing sites, and dynamic load balancing protocols that update and reorganize the distribution of workload at runtime. Finally, we analyze the dynamic load balancing protocols in terms of efficiency and scalability criteria.

1 Introduction

In the PARADISER (PARA*llel* and DIS*tributed* E*nvironment* for R*ules*) project, we are developing a complete database rule processing environment for expert database construction in a parallel and distributed setting. PARULEL (PA*rallel* RULE L*anguage*), the kernel language supported by this environment, has set-oriented execution semantics. It provides language features that support negation, aggregation, external functions, objects, and flexible control structures for manipulating large amounts of disk-resident data. It computes in cycles, consisting of *match*, *programmable conflict resolution*, and *fire* phases. PARADISER provides the infrastructure that allows PARULEL programs to be compiled,

*This work has been supported in part by the New York State Science and Technology Foundation through the Center for Advanced Technology under contract NYSSTF-CU01207901, and in part by NSF CISE grant CDA-90-24735.

distributed, and evaluated efficiently [Stolfo *et al.*, 1991; Dewan *et al.*, 1992; Wolfson *et al.*, 1991]. Our earlier experiences on the DADO project [Stolfo, 1984; Stolfo *et al.*, 1985] showed that under various schemes for evaluating rules in parallel, there is a large variance in runtime for the various distributed tasks. This issue has not had much treatment in the literature on parallel processing of rule programs. This paper is concerned with the details of the distributed evaluation schemes in PARADISER, and associated static and dynamic load balancing protocols.

The processing of a set of rules in a distributed expert database system may be either asynchronous or synchronous. In the asynchronous model, the sites may be viewed as independent threads of execution. When a thread interferes with another thread by removing facts used by it at an earlier inference cycle, the competing threads must be synchronized to maintain consistent operation [Dewan *et al.*, 1992; Wolfson *et al.*, 1991]. We have previously shown that this may be very expensive in the general case [Ohsie *et al.*, 1993]. To avoid these problems, PARADISER synchronizes the sites at each cycle boundary, exchanging new information as needed [1]. Another key design decision involves whether to use a replicated or a distributed (fragmented) database. While a distributed evaluation scheme should ideally use a distributed database, this may place very high demands on the underlying communication channels due to the reorganization of the distributed database for load balancing among the sites at runtime. Moreover, the key problems can be studied in either situation. Thus, for simplicity, the current implementation uses a fully replicated database. We have, however, made substantial progress toward accommodating distributed databases, details of which constitute the subject matter of a separate report [Dewan and Stolfo, 1993].

In PARADISER, *copy-and-constrain* [Stolfo *et al.*, 1985] is used for both compile-time workload distribution, as well as reorganizing the distributed computation at runtime for balanced operation. Program *replicas* are produced that are constrained to match smaller, disjoint subsets of the original database, reducing processing effort at each site. To alleviate the inefficiencies introduced by synchronous operation, we seek to minimize the variance in runtime among the sites at each cycle. This is achieved by keeping statistics on the distribution of data in the base relations. Initially, we partition the workload evenly by copying and constraining program replicas by using "restriction predicates", or simply "restrictions", attached to the rules. Each rule in a program replica contains a restriction on an attribute of some relation appearing on its left hand side. This attribute is called the *restriction attribute* (RA) for the rule. The restrictions limit the amount of data inspected by each rule, and hence by the entire program replica. The *Isoweight Copy-and-Constrain* (ICC) algorithm, outlined in Section 4.1, makes use of compile-time statistics to derive initial restrictions for the rules. At runtime, adjustments are made to the restrictions so that they exhibit balanced operation on subsequent cycles when the initial processing allocation fails to attain a balanced load. This is achieved via the *Dynamic*

[1] The updates are performed using a *lazy* protocol, making this apparent barrier synchronization point non-strict.

Restriction Adjustment (DRA) algorithm, which is outlined in Section 4.2.

The DRA algorithm is applied as part of a dynamic load balancing (DLB) protocol for detecting and correcting an unbalanced system. The particular DLB protocol for detecting imbalance and the DRA algorithm for correcting it constitute overhead not paid in the sequential case. The former of these sources deserves attention in that as the number of sites is scaled up, the communication costs for determining imbalance of the ensemble can display fast growth, and may dominate the overhead costs. Thus, the choice of DLB protocol is crucial for minimizing this overhead.

We compare various DLB protocols by analyzing their "isoefficiency" functions. Isoefficiency is a formalism proposed in [Kumar and Rao, 1987], and further studied in [Kumar *et al.*, 1991], that provides a measure of scalability of an algorithm-architecture combination. The isoefficiency analysis reported here is guiding our design decisions in the PARADISER environment.

The rest of this paper is organized as follows. Section 2 contains basic definitions. Section 3 discusses the copy-and-constrain technique and presents heuristics for choosing restriction attributes. In Section 4, we outline the ICC and DRA algorithms for database partitioning and provide an example of their application. Section 5 introduces a collection of dynamic load balancing protocols. A framework for analyzing these protocols is given in Section 6, and analysis of the protocols is given in Section 7.

2 Basic Definitions and Terminology

Rule-based expert systems typically follow the *Production System* (PS) model, consisting of a set of rules and a database of facts (working memory or WM). A rule consists of a left hand side (LHS), which is a conjunction of condition elements or "patterns" that may match elements of the WM. The right hand side (RHS) of a rule consists of actions that modify the database. The actions may be performed if the LHS is satisfied, thus producing a rule *instance*. The operational semantics of conventional rule languages such as OPS5 calls for the selection of a single instance from the set of instances by applying some selection strategy, also known as the *conflict resolution* method, and executing the actions associated with that instance. This procedure is carried out in a cycle consisting of the well-known *match-select-act* phases, until a predefined termination condition (usually a fixpoint) is reached. In the relational model, WM can be viewed as a relational database, and both the LHS and RHS of rules may be expressed as database queries. To the extent that the relational model can be mapped onto databases formed from collections of persistent objects, the same ideas apply to object-oriented databases as well. The LHS translates into joins and selections over the corresponding relations of the database that map the working memory in question [Hanson and Widom, 1992].

In rule-based systems, the stored facts define the extensional database (EDB), while the rules define the "derived" or intensional database (IDB). The IDB may be regarded as a view on the database, and this view can be either computed dynamically as needed, or stored in some fashion. The rules may refer

to any combination of EDB or IDB relations. In our work, "knowledge-base systems" refer loosely to systems consisting of a rule-based program together with an underlying database of facts about the domain. The rules encode domain-knowledge, and a well-defined operational semantics enables inference of new knowledge (the IDB) from the existing body of facts (the EDB).

3 Load Distribution by *Copy-and-Constrain*

In PARADISER, workload is distributed by dividing the effort involved in rule matching among the processing sites through the use of the copy-and-constrain paradigm [Stolfo *et al.*, 1985; Wolfson and Ozeri, 1990]. In practice, rules are replicated with additional constraints attached to each copy. Such restricted rules can be matched in parallel, thus providing a speedup [Stolfo *et al.*, 1985; Pasik, 1987]. To illustrate, let rule r be:

$$r: \; C_1 \wedge C_2 \wedge \cdots \wedge C_n \rightarrow A_1, \cdots, A_m \qquad n, m \geq 1$$

where the $C_i, i = 1 \cdots n$ are conditions *selecting* a subset of WM, and $A_j, j = 1 \cdots m$ is a set of m actions (updates to the database). Let $R(C_i)$ denote the tuples selected by C_i. The work, $W(r)$, to process r is bounded by the size of the cross product of the tuples selected on the LHS of r:

$$W(r) \leq \mid R(C_1) \mid \times \mid R(C_2) \mid \times \cdots \times \mid R(C_n) \mid$$

Suppose we choose to copy-and-constrain rule r on condition C_i to produce k new conditions $\{C_i^1, C_i^2, \cdots, C_i^k\}$, and k new replicas of r, $\{r^1, r^2, \cdots, r^k\}$, where each replica has C_i replaced by C_i^l, $l = 1 \cdots k$, on the LHS. If the new conditions are chosen such that $\mid R(C_i^l) \mid \approx \frac{\mid R(C_i) \mid}{k}$, $l = 1 \cdots k$, $\bigcap_{l=1}^{k} R(C_i^l) = \emptyset$, and $\bigcup_{l=1}^{k} R(C_i^l) = R(C_i)$, then

$$W(r^1) \approx \cdots \approx W(r^k) < \approx \mid R(C_1) \mid \times \cdots \times (\frac{\mid R(C_i) \mid}{k}) \times \cdots \times \mid R(C_n) \mid \approx \frac{W(r)}{k}$$

For appropriately chosen conditions, each of the k replicas require $\frac{1}{k}th$ the amount of work as the original rule r to process, forming the same set of instantiations. If the replicas are processed in parallel, the evaluation of r has been sped up by a factor of k.

3.1 Specifying Constraints

Four principal methods may be used for deriving constraints (the C_i in our previous example) to restrict the data matched by a rule under the copy-and-constrain paradigm. In the first case, if a free variable x in a rule condition corresponds to the restriction attribute (RA), and x takes values from a finite enumerable set of constants $\mathcal{D}$, we may construct the replicas by restricting x to subsets of $\mathcal{D}$. Secondly, if there is an easily computable hash function on the RA, we can apply hash partitioning. The third method applies if the RA has a

totally ordered domain. We can then simply divide the range of the RA directly into disjoint subranges. The fourth scheme, which generalizes the preceding three, is based upon data clustering. Here, a rule is constrained by restricting it to fragments of a base relation R. The fragments are constructed on the basis of a distance metric defined for the tuples of R, by forming *minimum distance* clusters around several *centroid tuples*. For simplicity, we restrict our discussion in this paper to the case where the RA draws its values from an ordered set, either directly or through a simple mapping, allowing restrictions to be expressed as simple arithmetic predicates. However, the techniques we develop carry over to the general case as well.

3.2 Heuristics for Choosing Restriction Attributes

To balance the load of distributed rule processing, we need to estimate the amount of computation required to execute a rule program against a particular database. However, the database in question may not be known at compile time. For example, the IDB distribution is not known initially. (In special cases, it may be possible to estimate the size of the IDB. See [Wolfson *et al.*, 1993]). Moreover, the EDB relations often cannot be exhaustively analyzed, since they may be very large. This suggests a heuristic approach for choosing RAs. The heuristic process we use for identifying the RA for a rule r proceeds in two phases. In the first phase, a set of possible RAs, called the *RA Candidate Set*, is identified. Suitable RA candidates are determined on the basis of the degree of uniformity of the distribution of the attribute value, as well as the size of the underlying relation. In the second phase, the candidate set is reduced to a single chosen restriction attribute. Space limitations do not permit a full description of the process in this paper. The reader is encouraged to see [Dewan and Stolfo, 1993] for details.

4 Dynamic Reorganization in PARADISER

In PARADISER, statistics kept in system catalogs play a key role in workload partitioning for distributed evaluation of general rule programs. The catalogs maintain the frequency distribution of tuples according to the value of the RA for a given relation. This is called the discrete frequency distribution (DFD), and is maintained as a discrete histogram. The DFD can be used to derive restriction predicates that select a subset of the tuples of a relation according to any desired workload schedule, called *loading weights*. A sophisticated facility creates the DFDs at compile time, and automatically maintains them using triggers as the database changes at runtime.

4.1 The Isoweight Copy-and-Constrain (ICC) Algorithm

The procedure for deriving restrictions is to compute subranges such that each subrange contains a desired fraction of the total number of tuples of the relation from which the restriction attribute is drawn. This uses a technique called

Weighted Range Splitting (WRS). WRS computes subranges so that the number of tuples with respect to the DFD in each subrange is proportional to the corresponding desired loading weight. Given the loading weights $\eta_i, i = 1 \cdots P$, for a set of P processors, these subranges can be computed in a single scan of a data structure that contains the DFD. In ICC, the loading weights are initially chosen to be all equal, i.e., $\eta_i = \frac{1}{P}, i = 1 \cdots P$. In contrast, DRA adjusts these weights dynamically by observing the performance of individual processing sites during system operation.

4.2 The Dynamic Restriction Adjustment (DRA) Algorithm

A dynamic load balancing protocol may consider all the sites in the ensemble of processors, or independent *groups* of sites from the ensemble. In general, a particular protocol may divide the ensemble into one or more groups prior to applying the strategies it encodes. Here, we outline the DRA algorithm, which computes the adjustments to the restriction predicates attached to the rules of the program versions in a group. The adjustment of the restriction predicate on a particular rule version r_S of some rule r at a site S is initiated only if the completion time of r_S differs from the average completion time of r over all sites in the group by more than some specified threshold.

In DRA, the restrictions on rule versions are adjusted using loading weights η_S^r computed at runtime for each rule r at site S, to divide the DFD by way of the WRS technique. While ICC initially divides the DFD of items equally among the sites, DRA uses loading weights η_S^r that are computed from observations of runtime performance. The idea is to assign greater loading weights to faster sites, and vice versa.

4.3 An Example

As an example, consider a simple rule program $\Pi = \{r_1, r_2\}$, and two sites $\{S_1, S_2\}$ over which the processing of Π is to be distributed. Let the replicated database be as follows:

$$\{A(1), A(2), A(3), A(5), A(10), A(25), B(1), B(4), D(1), D(6)\}$$

Suppose that x in relation A of both rules is chosen as the RA. Then ICC may produce the following workload partition, as evident in the restriction predicates on the LHS of the rules:

$$\begin{aligned}
(S_1)r_1 : A(x), B(y), (1 \leq x \leq 3) &\rightarrow C(x, y) \\
(S_1)r_2 : A(x), D(y), (1 \leq x \leq 3) &\rightarrow E(x, y) \\
(S_2)r_1 : A(x), B(y), (3 < x \leq 25) &\rightarrow C(x, y) \\
(S_2)r_2 : A(x), D(y), (3 < x \leq 25) &\rightarrow E(x, y)
\end{aligned}$$

Note that each subrange specified by the restriction predicates includes the same number of tuples, and that ICC is sensitive to the nonuniformity of the distribution of the values over the domain span of $A.x$. Suppose unbalanced operation is detected by the dynamic load balancing protocol. Consequently, DRA is applied to compute adjustments to the restrictions. Suppose that S_1 and S_2 had

completion times t and $2t$ respectively. The DRA algorithm would adjust the restrictions so that S_1 now has twice as much work as S_2:

$$\begin{aligned}
(S_1)r_1 &: A(x), B(y), (1 \le x \le 5) \quad \rightarrow \quad C(x,y) \\
(S_1)r_2 &: A(x), D(y), (1 \le x \le 5) \quad \rightarrow \quad E(x,y) \\
(S_2)r_1 &: A(x), B(y), (5 < x \le 25) \quad \rightarrow \quad C(x,y) \\
(S_2)r_2 &: A(x), D(y), (5 < x \le 25) \quad \rightarrow \quad E(x,y)
\end{aligned}$$

5 Dynamic Load Balancing (DLB) Protocols

We now present a collection of candidate dynamic load balancing (DLB) protocols that may be used in the PARADISER runtime environment. The differences in the protocols lie in the manner in which the architecture is "logically partitioned" and the particular sites that participate and cooperate with each other to rebalance the load. We refer to the size of this "participatory collection" as the group size Γ.

Global Coordinator (GC) Protocol: The *Global Coordinator* (GC) protocol makes load balancing decisions based upon global information on the state of rule program processing at all sites. Hence, the group size $\Gamma = P$. We outline it as follows:

1. At every processing cycle, each site keeps track of its local program and *rule completion times*. Sites determine the globally maximum (MAX) and minimum (MIN) completion times of the program replicas among all of the P sites in $\log(P)$ steps using a tournament selection method. At most $\log(P)$ steps after the last site completes, the coordinator site (root of the tournament tree) receives the MAX and MIN values. This site computes the imbalance measure, $I = \mid MAX - MIN \mid$.
2. If $I > C$ (some threshold), the global coordinator broadcasts a BALANCE signal. All sites then send rule and program completion times to the coordinator. For each rule at each site, the coordinator computes adjustments to the restrictions using the DRA algorithm and sends these to each site, which are then incorporated into the local program replicas. The coordinator then broadcasts a CYCLE signal.
3. On receiving a broadcast CYCLE signal, each site begins a new processing cycle, beginning with the match phase.
4. Termination is detected by the coordinator when every site reaches a fixpoint.

Round Robin (RR) Protocol: The *Round Robin* (RR) protocol makes load balancing decisions for groups of size $\Gamma = 2$. Each processing site has a distinct "paired" site which changes in a *cyclic* fashion at each invocation of the load balancing algorithm. The imbalance is computed pairwise between a site and its paired "target", and balancing is initiated on groups of size $\Gamma = 2$ using the DRA algorithm. Global synchronization is still performed by a coordinator site which

uses a simple counting mechanism to determine in logarithmic time whether all sites have completed their balancing actions. The details of the protocol are omitted here due to space limitations.

Random Probing (RP) Protocol: As in RR, the *Random Probing* (RP) protocol makes load balancing decisions for groups of size $\Gamma = 2$. Each site has a "paired" site which changes in a *random* fashion at each invocation of the load balancing algorithm. In contrast, RR changes the pairing of processing sites in a deterministic fashion. In other respects, this protocol is similar to RR.

Log(P) Nearest Neighbor (LNN) Protocol: This protocol is similar to GC. However, each site confines its balancing actions to a group of $\log(P)$ sites, where the total number of sites is P. Thus, the group size is $\Gamma = \log(P)$. The protocol includes a simple means of partially shuffling the members of each group in a random fashion prior to each load balancing phase. A "READY" signal is sent from one "representative" site in each group to a coordinator. The coordinator can still broadcast to every site. This protocol has lower communication overhead than GC, but the quality of load balancing may not be as high.

Sqrt(P) Nearest Neighbor (SNN) Protocol: This protocol is similar to LNN. However, each site confines its balancing actions to a group of $\sqrt{P}$ sites, where the total number of sites is P. Thus, the group size is $\Gamma = \sqrt{P}$. In all other respects, it is the same as the LNN protocol.

6 Framework for Comparing the DLB Protocols

The relative merits of various dynamic load balancing strategies is difficult to judge, since the usual performance metrics used to evaluate such systems are sensitive to numerous parameters, e.g., network topology, CPU speed, speed of I/O channels, problem size and so on. Changing any one of these parameters tends to invalidate any conclusions that one might draw from samples of performance data. To circumvent these problems, the *isoefficiency* metric was introduced in [Kumar and Rao, 1987]. This measure relates problem size to the number of processors such that the efficiency of the parallel or distributed computing ensemble remains constant. Isoefficiency analysis aims to capture the effects of a particular algorithm-architecture combination in a single expression as a measure of *scalability*.

A parallel algorithm is scalable if the efficiency can be maintained at some fixed value (between 0 and 1) by increasing the problem size W as some function f of the number of processors P. For example, if the problem size W must grow exponentially with P to maintain a fixed value of efficiency, then the algorithm is ill-suited to that architecture, since to obtain reasonable speedups, the problem size must grow at an exponential rate with respect to the number of processors. A good measure of how well a particular algorithm performs on a given architecture is to examine how close to *linear* the function f is. Linear isoefficiency functions indicate algorithms that are "highly scalable" on a given architecture. It can be shown that $f = \Omega(P)$ is a lower bound on any

isoefficiency function. Hence, any isoefficiency function that is linear or close to linear (e.g., $f = O(P \log^c P)$ for some integer c) are highly desirable. In the following discussion, we develop the basic framework for analyzing the dynamic load balancing protocols we presented in the previous section in terms of the scalability criterion and isoefficiency, based on the work of Kumar et al. We assume a collection of P networked workstations as the computing architecture.

Definitions:

1. W: Problem size or the measure of computation to solve a problem instance.
2. P: Number of processors in the ensemble with "identical" characteristics.
3. U_c: Time taken for one unit of computational work. In the case of database rule processing, this may be taken to be a logical read of a page from disk for the purpose of matching a rule against the database using relational operations such as **select**, **project**, and **join**.
4. T_c: Sum over all processors of the time spent in computation to solve the problem exclusive of communication and other overheads.
5. T_P: Running (elapsed) time for a given problem on a P processor ensemble.
6. T_o: Sum over all processors of the time spent in overhead activities such as interprocessor communication, waiting for messages, etc.
7. S: The speedup, or gain in computation speed, by using P processors. It is given by $S = \frac{T_c}{T_P}$.
8. E: The efficiency of the distributed computing ensemble on a problem instance. Intuitively, it is a measure of the utilization of computing resources for the problem at hand.

$$E = \frac{S}{P} = \frac{T_c}{T_P * P} = \frac{T_c}{T_c + T_o} = \frac{1}{1 + \frac{T_o}{T_c}}$$

In order to perform isoefficiency analysis, we proceed as follows. Observe that contributions to the overhead can come from communication among the sites, time spent in waiting for messages or responses from other sites, and in starvation. We assume that communication costs subsume all other overhead costs (which is reasonable in the presence of disk-resident data that we fundamentally presume). In order for the distributed system to meet the isoefficiency criterion, we must have $E = \gamma$ for some constant γ. From the definition of E, this implies $\frac{T_c}{T_P * P} = \gamma$, i.e., $T_c \propto T_P * P$. Since $T_c = U_c * W$ and U_c is fixed, we must have $W \propto P * T_P$

Our approach to establishing an expression for W as a function of P is to derive a lower bound on T_P in terms of the number of processors P. In the next section we treat each of the dynamic load balancing protocols we have previously presented.

7 Isoefficiency Analysis

Global Coordinator (GC):
In this protocol $\log(P)$ steps are needed to compute the global maximum and minimum of the completion times. In the worst case, the computed value of imbalance I would initiate load balancing actions at the end of every cycle. Then, each site communicates local information (e.g., completion times) to the coordinator. The coordinator computes adjusted range restrictions and communicates them to each site in $O(P)$ steps, since at least 1 message is necessary for every site. Thus, the worst case lower bound on T_P, computed on the basis of the requirements imposed by the load balancing protocol, is $O(P)$. Then we can write:

$$W \propto P\,(\log(P) + P) \implies W = c_1 P\,\log(P) + c_2 P^2$$

where c_1, c_2 are arbitrary constants. If load balancing is seldom required, c_2 is expected to be small, and $W = O(P \log(P))$; otherwise, it is $O(P^2)$.

Round Robin (RR):
For this protocol, computing the target for each source site at every cycle is a $O(1)$ operation. Similarly, sending local completion time data to the target and receiving back the adjusted range information from the target is also done in $O(1)$ steps. The number of sites, "COUNT", that will enter a balancing phase can be communicated to the coordinator within $O(\log(P))$ steps from the time the last site has completed its match phase, using a tournament algorithm. Using the same technique, the coordinator can be informed that the last site has completed its balancing actions in $O(\log(P))$ steps. Thus, the coordinator waits $O(\log(P))$ time before it can broadcast a message signaling for the next cycle of rule processing to begin. If we assume that the sites are roughly matched, a lower bound on T_P is $O(\log(P))$. Hence, for some arbitrary constant c,

$$W \propto P\,(\log(P)) \implies W = cP\,\log(P)$$

Random Probing (RP):
For this protocol, computing the target for each site at every cycle requires an average case analysis for the number of random probes necessary before a probe is successful in finding a free target. It can be shown using a straightforward analysis [Kumar *et al.*, 1991] that $O(P \log(P))$ probes are needed. As in the case of RR, once the target has been chosen, sending local completion time data to the target and receiving back the adjusted range information from the target can be done in $O(1)$ steps. As in RR, the coordinator knows within $O(\log(P))$ steps from the time the last site has finished balancing actions that it can broadcast a signal to start a new cycle. Thus, the lower bound on T_P is given by the dominating term $O(P \log(P))$. Hence, for some arbitrary constant c,

$$W \propto P\,(P \log(P)) \implies W = cP^2\,\log(P)$$

Log(P) Nearest Neighbor (LNN):
Using an analysis similar to that for GC, but using a set of $\log(P)$ sites, we can

derive a lower bound on T_P as follows. The contribution to T_P from the balancing actions alone from a group of size $\log(P)$ is $O(\log\log(P)+\log(P))$. Ignoring the double logarithm, this contribution is $O(\log(P))$. A representative site from each group communicates the READY signal to the coordinator, which then broadcasts a message signaling the continuation of normal processing. Since there are $\frac{P}{\log(P)}$ representatives, the READY signal from all representatives can be detected by the coordinator in $O(\log(\frac{P}{\log(P)})) = O(\log(P))$ time. The broadcast of the CYCLE signal takes $O(1)$ time. Thus, the lower bound on T_P is $O(\log(P))$, and for some arbitrary constant c, we have

$$W \propto P\,(\log(P)) \implies W = cP\,\log(P)$$

$\sqrt{P}$ Nearest Neighbor (SNN):
Using an analysis similar to the one for LNN, we see that the contribution to T_P from the balancing actions alone from a group of size $\sqrt{P}$ is $O(\log(\sqrt{P}) + \sqrt{P})$. Asymptotically, this contribution is $O(\sqrt{P})$. A representative site from each group communicates the READY signal to the coordinator, which then broadcasts a CYCLE signal. Since there are $\frac{P}{\sqrt{P}} = \sqrt{P}$ representatives, the READY signal from all representatives can be detected by the coordinator in $O(\log\sqrt{P}) = O(\log(P))$ time. The broadcast of the READY signal takes $O(1)$ time. Thus, the lower bound on T_P is $O(\sqrt{P})$, and for some arbitrary constant c, we have

$$W \propto P\sqrt{P} \implies W = cP^{1.5}$$

8 Conclusion

The analysis indicates clearly that for large numbers of processors, which we realistically assume will be commonplace in the foreseeable future, system reorganization is best implemented by either the Round Robin (RR) or Log Nearest Neighbor (LNN) protocols. In either protocol, the effect of the constant factor c on scalability is clearly crucial. The frequency of imbalance is dependent upon how well the DRA algorithm behaves in practice. If DRA does a poor job of predicting the amount of computation required by a particular program replica at a processing site, or the computational load placed on a site by some external application, then dynamic reorganization will be frequent and costly. It will be therefore important to design a scalable system that minimizes these costs as much as possible. Thus, the RR or LNN protocols would be appropriate candidates for implementation.

In our future work, we aim to study the behavior of the ICC and DRA algorithms under realistic problem scenarios. The frequency of application of DRA depends upon the particular problem and the distribution of data. The heuristics employed in ICC and DRA are intended to provide a rapid decision in distributing the workload at runtime. The heuristics may not work well in certain cases. The types of problems which pose challenges for the methods we have developed will be the focus of our attention in future work.

References

[Dewan and Stolfo, 1993] H.M. Dewan and S.J. Stolfo. The Distributed Evaluation of Rules in PARADISER. Technical Report In Preparation, Department of Computer Science, Columbia University, May (expected) 1993.

[Dewan *et al.*, 1992] H. M. Dewan, D. Ohsie, S.J. Stolfo, O. Wolfson, and S. DaSilva. Incremental Database Rule Processing in PARADISER. *Journal of Intelligent Information Systems*, 1:2, October 1992.

[Hanson and Widom, 1992] E.N. Hanson and J. Widom. An Overview of Production Rules in Database Systems. Technical Report RJ 9023 (80483), IBM Research Division, October 1992.

[Kumar and Rao, 1987] V. Kumar and V.N. Rao. Parallel Depth-First Search, Part II: Analysis. *I. Journal of Parallel Programming*, 16(6):501–519, 1987.

[Kumar *et al.*, 1991] V. Kumar, A.Y. Grama, and V.N. Rao. Scalable Load Balancing Techniques for Parallel Computers. Technical Report TR 91-55, Department of Computer Science, University of Minn[illegible] September 1991.

[Ohsie *et al.*, 1993] D. Ohsie, H.M. Dewan, S.J. Stolfo, and S. DaSilva. Performance of Incremental Update in Database Rule Processing. February 1993. Submitted to the 19th VLDB Conference.

[Pasik, 1987] A. Pasik. Improving Production System Performance on Parallel Architectures by Creating Constrained Copies of Rules. Technical Report CUCS-313-87, Department of Computer Science, Columbia University, 1987.

[Stolfo *et al.*, 1985] S. Stolfo, D.P. Miranker, and R. Mills. A Simple Processing Scheme to Extract and Load Balance Implicit Parallelism in the Concurrent Match of Production Rules. In *Proc. of the AFIPS Symposium on Fifth Generation Computing*, 1985.

[Stolfo *et al.*, 1991] S. Stolfo, O. Wolfson, P. Chan, H. Dewan, L. Woodbury, J. Glazier, and D. Ohsie. PARULEL: Parallel Rule Processing Using Metarules for Redaction. *J. Parallel and Distrib. Computing*, 13-4:366–382, 1991.

[Stolfo, 1984] S. J. Stolfo. Five Parallel Algorithms for Production System Execution on the DADO Machine. In *Proc. AAAI Conf.* AAAI, 1984.

[Wolfson and Ozeri, 1990] O. Wolfson and A. Ozeri. A New Paradigm for Parallel and Distributed Rule-processing. In *Proc. ACM-SIGMOD*, 1990.

[Wolfson *et al.*, 1991] O. Wolfson, H. Dewan, S. Stolfo, and Y. Yemini. Incremental Evaluation of Rules and its Relationship to Parallelism. In *Proc. of the ACM-SIGMOD 1991, Intl. Conf. on the Management of Data*, 1991.

[Wolfson *et al.*, 1993] O. Wolfson, W. Zhang, H. Butani, A. Kawaguchi, and K. Mok. A Methodology for Evaluating Parallel Graph Algorithms and its Application to Single Source Reachability. In *To appear in proceedings of PDIS-93*, 1993.

Using Semantic Information for Processing Negation and Disjunction in Logic Programs*

Terry Gaasterland[1] and Jorge Lobo[2]

[1] Mathematics and Computer Science Division, Argonne National Laboratory, gaasterland@mcs.anl.gov

[2] Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, jorge@eecs.uic.edu

Abstract. There are many applications in which integrity constraints can play an important role. An example is the semantic query optimization method developed by Chakravarthy, Grant, and Minker for definite deductive databases. They use integrity constraints during query processing to prevent the exploration of search space that is bound to fail. In this paper, we generalize the semantic query optimization method to apply to negated atoms. The generalized method is referred to as *semantic compilation*. We show that semantic compilation provides an alternative search space for negative query literals. We also how semantic compilation can be used to transform a disjunctive database with or without functions and denial constraints without negation into a new disjunctive database that complies with the integrity constraints.

1 Introduction

There are many applications in which integrity constraints can play an important role. For example, before expanding a database query, it is possible to incorporate the integrity constraints into the query to obtain a semantically equivalent query. The new query contains constraint information that prevents the exploration of search space that is bound to fail. In [CGM90], a semantic query optimization method is described that compiles integrity constraints into queries based on common positive literals that occur in both the constraint and the query. Database rules are optimized at compile time in order to minimize the amount of computation that must be done at run time. If queries are known at compile time, the entire query can be optimized. Otherwise, any remaining optimization is done at run time. Based on the same principles, integrity constraints can be also used to generate cooperative and informative answers and model user needs (see [MG90, Gaa92]).

We generalize the semantic query optimization method to apply to negated atoms. The generalized method is referred to as *semantic compilation*. This exploration has led to two significant results. The first result affords a new method

* Terry Gaasterland was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and Jorge Lobo under NSF grant #IRI-9210220.

to process negative query literals. Semantic compilation provides an alternative search space for negative query literals. Traversing the alternative search space can find answers in cases for which negation-as-finite-failure and constructive negation may not. To use the negation-as-finite-failure method to expand a negative query literal, the negative literal must be ground, and the search space of the positive ground atom must be finite. To use constructive negation effectively, the literal does not need to be ground but the search space must still be finite. Semantic compilation with negation as described in this paper allows correct answers to be found for negative literals that are neither ground nor finite. However, unless the integrity constraints completely describe all allowed states of the database, the answers for the negative literal may not be complete.

The second result applies to disjunctive deductive databases. Minker and Fernandez [FM91] noticed that there are many situations that can be represented directly with a set of disjunctive statements and a set of constraints that restrict the statements. To answer queries over such databases Fernandez and Minker generate the minimal models of the database and use the constraints to select the models that give the semantics of the database. Their algorithms work for stratified, function-free databases.

We show how for positive disjunctive deductive databases with or without functions and denial constraints without negation, semantic compilation can be adapted to avoid the construction of the minimal models. Other advantages of this approach are that minimal models do not have to be finite, and we can use any proof procedure for disjunctive databases to evaluate queries.

The next section gives background definitions. Section 3 describes and analyzes the method for handling negative literals with semantic compilation. Section 4 describes the compilation of constraints into disjunctive databases.

2 Background

Deductive databases are comprised of syntactic information and semantic information. The syntactic information consists of an *intensional database* (IDB) which is a set of clauses (or rules) of the form $A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n$, $m, n > 0$, where each A_i is an atom and each B_i is a literal (i.e. an atom or its negation), and the *extensional database* (EDB) which is the set of clauses of the form $A_1 \vee \cdots \vee A_m \leftarrow$. IDB clauses are also called rules. EDB clauses are also called facts. If all the B_is in the IDB rules are positive literals and $m = 1$, then the deductive database is called *definite*. If m is larger than 1 in at least one of the clauses, the database is called disjunctive. A pair of literals is said to be a pair of *complementary* literals if one is an atom and the other a negated atom.

The semantic information about the database consists of a set of integrity constraints (IC). The constraints considered in this paper have the form $\leftarrow C_1, \ldots, C_n, E_1, \ldots, E_m$ where each C_i is a literal whose predicate appears in an EDB fact or the head of an IDB rule and each E_i is an evaluable expression.

An integrity constraint restricts the states that a database can take. For example, the integrity constraint "No person can be both male and female." —

person(X),male(X),female(X), restricts people in a database from having both properties. Because the constraints on a database do not enable the deduction of new answers but rather give information about existing knowledge and answers, they are considered semantic information rather than syntactic information.

Chakravarthy, Grant, and Minker [CGM90] have shown how integrity constraints can be compiled into a query to identify space that is bound to fail. We now describe their algorithm. The algorithm assumes that the EDB and IDB contain function-free definite clauses, and that the IC is a finite set of first order clauses that always hold in the theory EDB∪IDB. It is assumed that the set IC contains constraints that can be derived from other constraints in IC.

The semantic optimization process performs two preliminary steps: *flattening* of the IDB clauses so that all IDB clause bodies contain only EDB predicates or recursive predicates and *variable substitution* of the IC clauses. Flattening is performed according to [Rei78] and consists of a series of unfoldings of the rules in the IDB. A database with direct recursion can be flattened by flattening the nonrecursive predicates in the recursive rules before merging the rules with the constraints [GGL+93]. Partial flattening can be done in rules with nondirect recursion by inspecting the dependency graph associated with the rules [GGL+93]. Without loss of generality, but possibly with some loss of potential query processing efficiency at run time, the flattening process does not have to be complete. Moreover, the negative literals are not flattened.

Background definition: Variable substitution

A variable substituted form of an integrity constraint is a clause obtained by

1. *Replacing each distinct constant in the constraint by a distinct variable and adding the fact that the constant equals the variable.*
2. *Replacing each variable that occurs more than once by new distinct variables, and adding equalities that represent the bindings.*

After preprocessing the IDB and the IC using variable substitution, the next step in semantic optimization is to find all partial constraints that apply to each IDB clause. A residue is obtained by determining whether part of some clause in IC partially subsumes an IDB clause. If so, then the remaining part of the IC clause becomes a residue that constrains the IDB clause – the residue expresses something that must be true when the clause is true. The general form of the integrity constraints that was produced by variable substitution allows us to find all possible residues.

Background definition: Partial subsumption

A (variable substituted) integrity constraint I partially subsumes a rule A if a subset of I subsumes A but I does not subsume A. An integrity constraint $I = \leftarrow C_1, \ldots, C_n$ partially subsumes a rule $R = A \leftarrow B_1, B_2, \ldots, B_m$, if and only if there exists a nonempty subset S of $\{C_1, \ldots, C_n\}$ and a substitution θ such that $S\theta \subseteq \{B_1, \ldots, B_m\}$.

Background definition: Residue

If $C_{i_1}, \ldots, C_{i_k}$ are the literals in the constraint that are not in S, then the clause $\leftarrow (C_{i_1}, \ldots, C_{i_k})\theta$ is the residue *obtained from R and I.*

Once a set of residues is found for an IDB clause, each residue may be incorporated into the clause. Attaching the residue to the IDB clause produces a semantically equivalent clause.

Background definition: Semantically constrained rules

A semantically constrained rule is defined to be of the form

$A \leftarrow B_1, \ldots, B_m, \{R_1, \ldots, R_n\}$.

Here A is the procedure head of the rule, each literal B_i is either a regular literal or an evaluable predicate, and the R_js are residues obtained from the rule $A \leftarrow B_1, \ldots, B_m$ and the integrity constraints.

Background definition: Semantically constrained database

Let D be a definite database and let E, I, and C be the EDB, IDB, and ICs that form D. The semantically constrained database of D, D', is the set of semantically constrained rules obtained from D.

Consider a database consisting of two IDB clauses:

economic_ties(C1,C2) ← trade(C1,Z), trade(Z,C2)

trade(C1,C2) ← friends(C1,C2)

meaning *C1 has economic ties with C2 if C1 trades with some Z that trades with C2* and *C1 trades with C2 if C1 is friends with C2*. The variable substituted integrity constraint above partially subsumes the first IDB clause to produce the residue $\{\leftarrow C1{=}usa,\ C2{=}iraq\}$.

A new semantically constrained clause is produced when the residue is merged with the IDB clause

economic_ties(C1,C2) ← trade(C1,Z), trade(Z,C2), {← C1=usa, C2=iraq}

The constrained axiom says that it is not possible for the countries $C1$ and $C2$ to have economic ties if $C1$ is *usa* and $C2$ is *iraq*.

The semantically constrained database consists of the semantically constrained clauses together with the EDB facts and the integrity constraint above.

3 Handling Negative Literals

The semantic optimization method [CGM90] augments queries and rules with semantic information about the search space of positive atoms in a query or the rules when those atoms occur positively in the body of one or more integrity constraints. In this section, we generalize the approach so that it provides semantic information about the search space of negative as well as positive literals in the query. The generalized method handles all cases in which literals with the same predicate occur positively or negatively in one or more constraints. Semantic information can be obtained in four cases:

1. When a predicate symbol of a positive constraint literal appears in a positive literal in the query or the rules.
2. When a predicate symbol of a negative constraint literal appears in a negative literal in the query or the rules.
3. When a predicate symbol of a negative constraint literal appears in a positive literal in the query or the rules.

4. When a predicate symbol of a positive constraint literal appears in a negative literal in the query or the rules.

Case 1 is covered in [CGM90] and forms the basis for the treatment of Cases 2, 3 and 4. Case 2 can be treated like Case 1 by temporarily renaming the negated predicates as unique new positive predicates and applying semantic compilation as in case 1. Case 3 reduces to using the integrity constraint as a deductive rule and is not particularly interesting. Case 4 is the most interesting: semantic compilation with negation provides a means to find answers for queries that contain possibly nonground negative literals, in which case regular procedures such as SLDNF-resolution do not apply, and/or possibly an infinite number of answers, in which case approaches such as constructive negation [Cha88, Prz89b] cannot be used.

3.1 Algorithm for Generalized Semantic Compilation

Semantically compiling a set of integrity constraints with a database rule or query has two main steps: (1) obtaining partial constraints, and (2) integrating the partial constraints into the body of the query or rule. A query may be regarded as a rule of the form $Query \leftarrow Q_1, \ldots, Q_n$. From now on, we will refer to both queries and rules as axioms. To compile an integrity constraint into an axiom based on the correspondence of a positive or negative axiom literal and a positive or negative integrity constraint literal, we define *mixed partial subsumption*. The term *mixed* refers to the fact that the correlation is done on two complementary literals, one from the constraint and one from the axiom. Since the residues produced by mixed partial subsumption contain a different type of information from the constraints found by regular partial subsumption, we call them *mixed residues*. We also define a merging algorithm that replaces the axiom literal with the disjunction of its mixed partial constraints.

Definition 1. Mixed partial subsumption
The mixed partial subsumption *of an axiom* $R = A \leftarrow B_1, B_2, \ldots, B_m$, *by an integrity constraint* $I = \leftarrow C_1, \ldots, C_n$, *can occur if and only if there exist a* B_i *and a* C_j *such that*

1. B_i *and* C_j *are complementary literals.*
2. *There is a substitution* θ *such that* $C_j\theta \equiv \neg B_i$.

The clause $(\leftarrow C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n)\theta$, *is the* mixed residue *obtained from* R *and* I.

The integrity constraint $\leftarrow C_1, \ldots, C_n$ can be interpreted to say that it is not the case that C_1, and C_2, ..., and C_n are simultaneously true in the database. In particular, if the constraint is restricted with the substitution θ, it is not the case that the conjunction $C_1\theta \wedge \cdots \wedge C_n\theta$ is true in the database. Hence, if the conjunction $C_1\theta \wedge \cdots \wedge C_{j-1}\theta \wedge C_{j+1}\theta \wedge \cdots \wedge C_n\theta$ is true, then it is not possible for $C_j\theta$ to be true. Since in deductive databases negation is interpreted as negation-as-failure, this is equivalent to saying that $\neg C_j\theta$ is true. In other words, we can rewrite the constraint to be $(\neg C_j\theta \leftarrow C_1\theta, \ldots, C_{j-1}\theta, C_{j+1}\theta,$

$\ldots, C'_n\theta$). But notice that $\neg C'_j\theta \equiv B_i$. Therefore, we have ($B_i \leftarrow C'_1\theta, \ldots, C'_{j-1}\theta, C'_{j+1}\theta, \ldots, C'_n\theta$). The mixed residue obtained through mixed partial subsumption defines a search space that is an alternative to the search space of B_i. This new implication deduced from the semantic information motivates the following definition of merging a rule with an integrity constraint.

Definition 2. Mixed merging

To merge the mixed residue $(C'_1, \ldots, C'_{j-1}, C'_{j+1}, \ldots, C'_n)\theta$ *with the rule* R, *replace the rule by the new rule*

$A \leftarrow B_1, B_2, \ldots, B_{i-1}, C'_1\theta, \ldots, C'_{j-1}\theta, C'_{j+1}\theta, \ldots, C'_n\theta, B_{i+1}, \ldots, B_m$.

Algorithm 1. Semantic Compilation with Negations

Let DB be an initial deductive database. Let DB' contain only the EDB of DB. Then,

1. *For each integrity constraint in the database, use mixed partial subsumption to find all mixed residues between the integrity constraint and the axiom.*
2. *For each literal in the axiom that has a corresponding mixed residue, use mixed merging to replace the literal with the mixed residue.*
3. *For each integrity constraint in the database, use partial subsumption to find residues for the new axiom and the original flattened axiom in DB.*
4. *Add all regular residues to the new axiom and the original axiom using the usual merge definition. Add the new axioms to DB'.*

DB' contains the new semantically compiled database.

This algorithm performs semantic compilation for cases 1, 3, and 4 above. The following example illustrates the algorithm.

The database below describes two disjoint groups of points connected by edges (i.e., two distinct connected components of the graph). The disjointness can be described by an integrity constraint. The predicate *reachable* computes all ordered pairs that are connected by an edge. The predicate *unreachable* has a search tree with a cycle on *reachable*. IC1 is sufficient to describe the disjoint edge groups. The database consists of the following clauses:

```
% group 1:          % group 2:          reachable(X,Y) ← edge(X,Y).
edge(a,b).          edge(c,d).         reachable(X,Y) ← reachable(X,Z),edge(Z,Y).
edge(b,a).          edge(d,c).         unreachable(X,Y) ← ¬reachable(X,Y)
IC1 ← reachable(a,W1),reachable(c,W1).
```

Variable substitution of the integrity constraint produces a new form of the constraint: $\leftarrow$ *reachable(A,W1),reachable(C,W2), A=a,C=c,W1=W2.* Mixed partial subsumption of IC1 with the recursive rule for *unreachable* produces the following set of residues: { $\leftarrow$ *reachable(C,X2),X=a,C=c,Y=X2.*, $\leftarrow$ *reachable(A,X1),X=c,A=a,Y=X1* } Merging the residues with the rule produces the following new rule:

unreachable(X,Y) $\leftarrow$ *((¬reachable(X,Y)) ∨ (reachable(C,X2),X=a,C=c,Y=X2) ∨ (reachable(A,X1),A=a,X=c,Y=X1)).*

Unfolding the semantically compiled rule gives three new rules:

R1 *unreachable(X,Y)* $\leftarrow$ *¬reachable(X,Y).*

R2 *unreachable(X,Y)* $\leftarrow$ *reachable(C,X2),X=a,C=c,Y=X2.*

R3 *unreachable(X,Y)* $\leftarrow$ *reachable(A,X1),X=c,A=a,Y=X1.*

R2 and R3 can be reduced to the following:

R2′ $unreachable(a,Y) \leftarrow reachable(c,Y)$ R3′ $unreachable(c,Y) \leftarrow reachable(c,Y)$

From R1, which contains a negative literal, the query $\leftarrow unreachable(X,Y)$ cannot be answered using a procedure such as SLDNF-resolution since the nonground query $\leftarrow \neg reachable(X,Y)$ has to be solved. One possibility is to extend the proof procedure with constructive negation. However, to use constructive negation, one must carefully extend SLDNF-resolution to incorporate constructive answers. Even though there is a finite number of answers for the query $\leftarrow reachable(X,Y)$, a simple application of SLD-resolution to obtain the answers will result in an infinite computation. However, with R2′ and R3′, produced by Algorithm 1, two answer substitutions can be computed: $\{X/a, Y/d\}$ and $\{X/a, Y/c\}$. Moreover, suppose the integrity constraint $\leftarrow reachable(a,W1), reachable(c,W2), reachable(W1,W2)$ is also added to the database. Then, after the compilation, the following two rules are generated:

R4 $unreachable(X,Y) \leftarrow reachable(a,X), reachable(c,Y)$

R5 $unreachable(Y,X) \leftarrow reachable(a,X), reachable(c,Y)$.

From these two rules the rest of the answers for $unreachable(X,Y)$ are obtained.

The following theorem formally establishes the correctness of Algorithm 1.

Theorem 1. *Let the tuple DB = (IDB,EDB,IC) be a deductive database. Let DB′ = (IDB′,EDB,IC) be the database obtained after applying Algorithm 1. Let Q be a query. A substitution θ is a correct answer substitution for DB ∪ {Q} if and only if θ is a correct answer substitution for DB′ ∪ {Q}.*

3.2 Comparison with Constructive Negation

When a query contains a negative literal, there are two procedural methods for expanding the search space of the negative literal. Negation-as-finite-failure (NAFF), used in SLDNF-resolution, expands the entire search tree for the positive atom in the literal. *If the entire tree fails finitely, then the negative atom* can be considered to be true. NAFF requires the negative literal to be ground and for the positive atom's search tree to be finite. Otherwise, it can return no answers. Constructive negation eliminates the need for the negative literal to be grounded. It expands the search tree of the literal's positive atom to obtain all the answer substitutions for the variables in the atom and constructs an expression formed with equalities and inequalities that states values that the variables in the literal can take or cannot take. Basically, any set of values that make the atom in the literal true are disallowed as substitutions for the negated atom. Constructive negation still has the limitation that the search tree associated with the positive atom must be finite. Another disadvantage is that constructive negation requires all the possible answers for the positive atom before it can return the answer for the negation of the atom. This restriction can make the process very long and extremely complex, especially if a single answer suffices the user posing the query to the database.

As in constructive negation, Algorithm 1 enables a search for values of variables in nonground negative literals. It uses constraint information to identify search space that cannot co-occur with the query's negated literal. The algorithm

produces a new semantically equivalent query that covers the search space and constructs substitutions of allowed values for the variables that appear in the negative literal. Unlike constructive negation, the set of substitutions obtained using the algorithm is not necessarily complete. However, the answers that are found are correct. For example, consider the following query, constraint and residue: Q ← ¬*P(X,Y,Z)*,*S(Y)*. IC ← *P(X,Y,Z)*, *R(X,Z)*. Residue: {*R(X,Z)*}
Suppose the residue *R(X,Z)* is expanded as a query and produces the answer substitutions #1 = {a/X, b/Z} and #2 = {aa/X, bb/Z}. The constraint says that no state of the database will contain both *R(X,Z)#1* and *P(X,Y,Z)#1* or both *R(X,Z)#2* and *P(X,Y,Z)#2*. Thus, any bindings that are obtained for *Y* through *S(Y)* can be concatenated with the #1 and #2 to produce answer substitutions for the query. Suppose expansion of *S(Y)* produces two substitutions, {c/Y} and {cc/Y}. Then the final set of answer expressions for the query is the following: {a/X, c/Y, b/Z}, {aa/X, cc/Y, bb/Z}, {a/X, cc/Y, b/Z}, {aa/X, c/Y, bb/Z}. Moreover, in a Prolog-like interpreter, the answers are given one at a time, and if only one answer is required, the computation process can be stopped after the first answer is obtained. Other answers for *R(X,Z)* may exist.

4 Disjunctive Databases and Constraints

When a set of integrity constraints IC is associated with a disjunctive database P, the semantics of (P + IC) can be described in two basic ways. We consider the set of models associated with P alone:

1. If each constraint is true in each model, then the database is considered consistent with IC, and together the models give *the meaning* of the database. Otherwise, the database is considered inconsistent with IC, and the database has no meaning.
2. If there is some model in which all the constraints are true, then the database is considered to be consistent with IC. The set of models in which all constraints are true describes the meaning of the database. Otherwise, the database is considered inconsistent with IC, and the database has no meaning.

For definite databases, under minimal model semantics, these two cases are equivalent because only one model is associated with a database: its minimal model. However, under *minimal model semantics*, disjunctive databases may be inconsistent with IC by the first definition yet be consistent by the second definition. This is possible because such databases have multiple minimal models.

The following disjunctive database illustrates this point:

IDB1: *covalent_bond(A,B)* ∨ *double_bond(A,B)* ← *carbon(A)*, *bonded(A,B)*.

EDB1: *carbon(item1)*. EDB2: *hydroxyl(item2)*. EDB3: *bonded(item1,item2)*.

The rule IDB1 says, *If some entity A is a carbon atom and it is bonded to some other entity B, then the bond is either a covalent bond or a double bond.* The facts indicate that *item1* is a carbon atom, *item2* is a hydroxyl group, and the two items are bonded. The database has two minimal models:

M_1 = { carbon(item1), hydroxyl(item2), bonded(item1,item2), double_bonded(item1,item_n), }

M_2 = { carbon(item1), hydroxyl(item2), bonded(item1,item2), covalent_bond(item1,item_n), }

Now consider a constraint that says, *Nothing can be a hydroxyl group and be double-bonded to something else:* IC1 $\leftarrow$ *double_bond(A,B), hydroxyl(B).* The first minimal model violates the constraint with the substitution {*item1/A, item2/B*}. Thus, according to the integrity constraint, the first minimal model should be disregarded when answering queries to the database.

Fernandez and Minker have defined an algorithm that takes a disjunctive database without function symbols, computes its minimal models, and eliminates the models that are not consistent with the set of constraints. Answers to queries are computed over these models [FM91]. An alternative way to give answers to queries that are consistent with the integrity constraints is to transform the database into a new database that incorporates the integrity constraint information. If all of the minimal models of the new database are consistent with the constraints, then any answers obtained through any disjunctive query processing mechanism would be consistent with the constraints as well.

We modify the definition of mixed partial subsumption and mixed merging in order to describe a procedure that performs such a transformation on (not necessarily function-free) disjunctive databases and constraints without negation.

Definition 3. Disjunctive subsumption

The disjunctive subsumption *of an axiom* $R = A_1 \vee \cdots \vee A_k \leftarrow B_1, B_2, \ldots, B_{n_r}$ *by an integrity constraint* $I = \leftarrow C_1, \ldots, C_n$, *can occur if and only if there exist a* A_i *and a* C_j *such that they unify with mgu* θ. *The clause* $(\leftarrow C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n)\theta$, *is the* disjunctive residue *obtained from* R *and* I.

Definition 4. Disjunctive merging

To merge the disjunctive residue $(\leftarrow C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n)\theta$ *with the rule* R, *replace the rule by the new rule* $(A_1 \vee \cdots \vee A_{i-1} \vee A_{i+1} \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_n)\theta$.

Procedure 1. Semantic Compilation in Disjunctive Databases

Repeat the following steps until no I *in* IC *disjunctive subsumes any rule in* P:

1. *Select* I *from* IC; *let* IC *be* $IC - \{I\}$.
2. *Let* P' *be* P.
3. *While* $P' \neq \emptyset$ *do*
   (a) *Use disjunctive subsumption to obtain all disjunctive residues between* I *and the axioms in* P'.
   (b) *Use disjunctive merging to generate the new disjunctive rules* NR.
   (c) *If the empty clause is in* NR *report that* P *is inconsistent with* IC *and stop.*
   (d) *Remove each rule with an empty head from* NR *and add it to* IC.
   (e) *Let* P *be* $P \cup NR$.
   (f) *Let* P' *be* NR.

Consider our disjunctive deductive database about a carbon atom and a hydroxyl group. When the procedure is applied to IC1 and EDB2, the first iteration produces the following intermediate constraint: IC2 $\leftarrow$ *double_bond(A,item2).* The

intermediate constraint is added to the database, and the procedure is applied again to produce a new IDB rule, IDB2, from IC2 and IDB1:

IDB2 $covalent_bond(A,item2) \leftarrow carbon(A),\ bonded(A,item2)$

After all the interactions, the final database produced by the procedure consists of the following rules and facts (note the addition of IDB3):

IDB1 $covalent_bond(A,B) \vee double_bond(A,B) \leftarrow carbon(A),\ bonded(A,B)$

IDB2 $covalent_bond(A,item2) \leftarrow carbon(A),\ bonded(A,item2)$

IDB3 $covalent_bond(A,B) \vee double_bond(A,B) \leftarrow carbon(A),\ bonded(A,B),\ \neg\ hydroxyl(B)$

EDB1 $carbon(item1)$ EDB2 $hydroxyl(item2)$ EDB3 $bonded(item1,item2)$

The new database has one minimal model: $\{carbon(item1),\ hydroxyl(item2),\ bonded(item1,item2),\ covalent_bond(item1,item2)\}$. The model is consistent with IC1.

Theorem 2. *Let P be a positive disjunctive database. Let IC be a set of integrity constraints. Let P' be a positive disjunctive database produced by Procedure 1 from P and IC. Then the following holds: (1) Any minimal model of P that satisfies IC is a model for P', and (2) Any minimal model of P' is a model for $P \cup IC$.*

We are not able to guarantee termination of Procedure 1 since it applies to all disjunctive databases and the question of whether or not a set of IC's is consistent with a set of disjunctive clauses is only semi-decidable. However, Procedure 1 is a step toward the extension of semantic compilation to disjunctive theories. We believe that with minor modifications, the same procedure can be used for more general constraints and databases. The modifications will depend on the semantics chosen for negation in disjunctive databases.

5 Conclusion

Chakravarthy, Grant, and Minker [CGM90] gave an effective method to optimize queries to definite deductive databases for cases in which partial subsumption involves positive atoms. We have generalized the method to apply to negative literals in either the database axioms or the integrity constraints or both. We have studied all possible syntactic interactions between the literals in the query and the literals in the integrity constraints: positive-positive, negative-negative and negative-positive.

The generalized method, called semantic compilation, enables answers to be found for queries with negative literals in cases for which constructive negation or SLDNF find no answers. We have shown that the answers obtained from a semantically compiled database are correct and, in some cases, complete.

When the generalized semantic compilation is applied to disjunctive databases that are inconsistent with the integrity constraints for some minimal models, a new database is produced whose minimal models are all consistent with the integrity constraints if that database exists. In this new database, queries can be answered using any procedure for disjunctive databases without requiring a special mechanism to check the constraints. It remains to be seen how semantic compilation can be applied to disjunctive databases with negation.

References

[CGM90] U.S. Chakravarthy, J. Grant, and J. Minker. Logic based approach to semantic query optimization. *ACM Transactions on Database Systems*, 5(2):162–207, June, 1990.

[Cha85] U. Chakravarthy. *Semantic Query Optimization in Deductive Databases*. Ph.D. thesis, University of Maryland, Department of Computer Science, College Park, 1985.

[Cha88] D. Chan. Constructive negation based on the completed databases. In R. A. Kowalski and K. A. Bowen, editors, *Proc. 5th International Conference and Symposium on Logic Programming*, pages 111–125, Seattle, Washington, MIT Press, August 15–19, 1988.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[FM91] J. A. Fernández and J. Minker. Bottom-up evaluation on disjunctive databases. In K. Furukawa, editor, *Proc. International Conference on Logic Programming*, pages 660–675, Cambridge, Massachusetts, 1991. MIT Press.

[Gaa92] T. Gaasterland. *Generating Cooperative Answers in Deductive Databases*. Ph.D. thesis, University of Maryland, Department of Computer Science, College Park, 1992. (Technical Report UMIACS-TR-92-107, CS-TR-2968, University of Maryland, October, 1992.)

[GGL+93] T. Gaasterland, M. Giuliano, A. Litcher, Y. Liu, and J. Minker. *Using integrity constraints to control search in knowledge base systems*. To appear, Journal of Expert Systems.

[GGMN92] T. Gaasterland, P. Godfrey, J. Minker, and L. Novik. A cooperative answering system. In Andrei Voronkov, editor, *Proceedings of the Logic Programming and Automated Reasoning Conference*, pages 101–120, volume 2, Russia, July 1992.

[GM78] H. Gallaire and J. Minker, editors. *Logic and Databases*. Plenum Press, New York, April 1978.

[Kow79] R. Kowalski. *Logic For Problem Solving*. Elsevier Science Publishing Co, Inc., Oxford, 1979.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, second edition, 1987.

[MG90] J. Minker and A. Gal. Producing cooperative answers in deductive databases. In P. Saint-Dizier and S. Szpakowics, editors, *Logic and Logic Grammar for Language Processing*. L.S. Horward, Ltd., 1990.

[Prz89b] T. C. Przymusinski. On constructive negation in logic programming. In E.L. Lusk and R.A. Overbeek, editors, *Proc. of the North American Conference of Logic Programming*, Cleveland, Ohio, October 16-20, 1989. Addendum to Proceedings.

[Rei78] R. Reiter. Deductive question answering on relational databases. In H. Gallaire J. Minker, editor, *Logic and Data Bases*, pages 149–177. Plenum Press, New York, 1978.

[Ros89] K. A. Ross. A procedural semantics for well founded negation in logic programs. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, Philadelphia, Pennsylvania, ACM Press, March, 29-31, 1989.

On the Interpretation of Set-Oriented Fuzzy Quantified Queries and their Evaluation in a Database Management System

Patrick BOSC & Ludovic LIETARD & Olivier PIVERT

IRISA/ENSSAT
BP 447
Lannion Cédex
FRANCE

Abstract. Many propositions to extend database management systems have been made in the last decade. Some of them aim at the support of a wider range of queries involving fuzzy predicates. Unfortunately, the evaluation of these queries is computationally complex and efficiency is considered in this paper. We focus on a particular subset of queries, namely those using fuzzy quantified predicates. More precisely, we will consider the case where such predicates apply to sets of elements. Based on some interesting properties of α-cuts of fuzzy sets, we are able to show that the evaluation of these queries can be significantly improved with respect to a *naïve strategy* performing exhaustive scans of sets.

1 Introduction

The database management systems currently available are based on the relational model and they suffer several limitations regarding user or application needs. In particular, it is assumed that data are precisely known (or fully unknown) and queries are based on crisp conditions. The notion of imprecision can be introduced in such systems at two levels: for representing imprecise or uncertain data and to allow flexible queries. In this paper, we will only consider the second aspect, that is to say that the data are assumed to take their values in ordinary universes, whereas queries may contain imprecise conditions.

In the context of an extended relational language supporting imprecise querying such as SQLf [1, 2], queries are viewed as fuzzy predicates. The query is associated to a threshold α and the retrieved data are the elements of the α-cut. In such a language, it seems natural to compose fuzzy predicates and to introduce fuzzy quantifiers inside queries. Various kinds of compound fuzzy predicates have been proposed in recent years [3, 6]. Base predicates described as fuzzy sets (i.e. by means of characteristic functions) can be altered by linguistic modifiers and arranged together using connectors or aggregates in order to reach the appropriate semantics. Fuzzy quantifiers were first introduced by L.A. Zadeh [8] to generalize the existential ($\exists$) and universal ($\forall$) quantifiers. Few years after, R. Yager [5, 6, 7] suggested another definition for fuzzy quantifiers and a new approach, based on possibility theory, was also proposed by H. Prade [4]. In this paper, we deal with the evaluation of fuzzy quantified predicates which concern sets of tuples. More precisely, our aim is to point out some efficient strategies for the evaluation of fuzzy quantified predicates (according to Zadeh's and Yager's interpretations only), since efficiency is a key point in DBMS's.

In section 2, fuzzy quantified predicates are introduced along with their interpretation according to Zadeh and Yager and their evaluation is presented in section 3. To conclude, we summarize the main results and draw some directions for future works.

2 The Quantification of Set-Oriented Fuzzy Predicates

2.1 Set-Oriented Fuzzy Predicates

In the usual relational framework it is possible to distinguish the predicates applying to individual elements (x's) of a set X and the predicates whose argument is a set X of elements. Two typical examples of these categories in an SQL language are (over the relation UNIVERSITY(teacher, department, salary)):

"*select* teacher *from* UNIVERSITY *where* salary > 4000"

and "*select* department *from* UNIVERSITY
group by department *having* avg(salary) > 4000".

It is possible to extend this notion to fuzzy predicates and to define a set-oriented fuzzy predicate which gives to set X a value ranked in [0,1]. Thus, in this paper, we will concentrate on the evaluation of this type of query:

"*select* ... *from* R *group by* Y *having* Q *are* A"

where Q is a fuzzy quantifier,Y an attribute of the relation R and A a set-oriented fuzzy predicate. Mainly, the topic of this article will be the expression and evaluation of the predicate "Q x's are A" where the x's are the elements of a usual set X. An example, over the same relation UNIVERSITY, could be:

"*select* α department *from* UNIVERSITY
group by department *having* many *are* well-paid".

This query corresponds to the sentence "find the departments, satisfying many of their teachers are well-paid, with an overall degree over α". Thus, for each department, we need to evaluate the set-oriented predicate "many teachers are well-paid" to determine if its value is over α.

2.2 Fuzzy Quantifiers According to Zadeh

Zadeh [8] distinguishes the absolute quantifiers and the relative ones. The absolute quantifiers (about 3, at least a dozen) are defined on a number (the absolute fuzzy cardinality) while the relative quantifiers (about one half, almost all) are defined on a proportion (the relative fuzzy cardinality). Thus, an absolute quantifier is represented by a function Q from [0,n] to [0,1] whereas a relative quantifier is represented by a function Q from [0,1] to [0,1]. In both cases the value Q(j) is the satisfaction the user gives to the quantification if j criteria are satisfied.

If Q_a stands for an absolute quantifier, the set oriented predicate "Q_a x's are A" is then interpreted according to the formula $Q_a(\Sigma_X \mu_A(x))$ where the x's are the elements of the set X. If Q_r stands for a relative quantifier, the set oriented predicate "Q_r x's are

A" is interpreted according to the formula $Q_r(\Sigma_X \mu_A(x)/n)$ where n is the usual cardinality of the set X.

Let's consider the absolute quantification "About 3 x's are A" and the relative quantification "At least half x's are A":

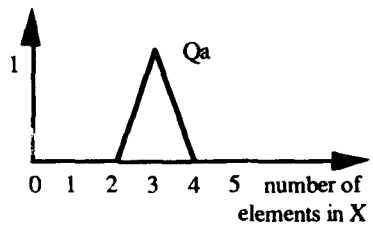


Fig. 1. The absolute quantifier: "About 3"

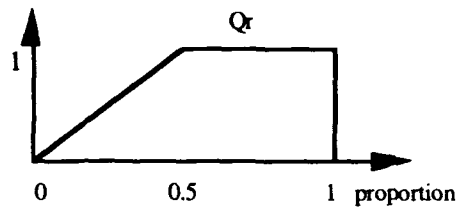


Fig. 2. The relative quantifier: "At least half"

The fuzzy set $\{0.5/x_1, 0.8/x_2, 1/x_3\}$ satisfies "About 3 x's are A" with a degree $Q_a(2.3) = 0.3$. It satisfies the predicate "At least half x's are A" with a degree $Qr(0.76) = 1$.

The interpretation of an absolute quantifier seems very difficult to apply in all cases. This is due to the fuzzy cardinality which may be equal for two sets, while these two sets are very different. As an example the set $\{0.1/x_1, 0.1/x_2, ..., 0.1/x_{30}\}$ is as "About 3" as the set $\{1/x_1, 1/x_2, 1/x_3\}$ but, actually, according to the human appreciation of "About 3", they are very different. Thus, this paper will only deal with the evaluation of relative quantifiers according to Zadeh's interpretation.

2.3 Fuzzy Quantifiers According to Yager

Yager [5, 6, 7] suggests we represent proportional (equivalent to increasing monotonic) quantifiers by means of an Ordered Weighted Averaging aggregation (OWA). First of all let us recall the definition of an OWA operator :

$$OWA(w_1, ... ,w_n,x_1, ..., x_n) = \sum_{i=1}^{n} (w_i * x_{k_i})$$ where x_{k_i} is the i^{th} largest value among the x_k's.

A proportional quantifier Q is defined as : $Q(0) = 0$, $\exists k$ such that $Q(k) = 1$ and $\forall$ a,b if $a > b$ then $Q(a) \geq Q(b)$. The interpretation of Yager is to compute the weigths w_i's of the aggregation from the function Q describing the quantifier. In case of an absolute proportional quantifier $w_i = Q(i) - Q(i-1)$ and in case of a relative proportional one, assuming n is the crisp cardinality of X, $w_i = Q(i/n) - Q((i-1)/n)$.

Let's consider the relative proportional quantifier "At least half " of figure 2. The fuzzy set $\{0.5/x_1, 0.8/x_2, 0.7/x_3\}$ satisfies "At least half x's are A" with a degree given by $OWA(w_1, w_2, w_3, 0.5, 0.8, 0.7)$. We compute:

$$w_1 = Qr(1/3)-Qr(0) = 2/3 \quad w_2 = Qr(2/3)-Qr(1/3) = 1/3 \quad w_3 = Qr(1)-Qr(2/3) = 0$$

$$OWA(w_1, w_2, w_3, 0.5, 0.8, 0.7) = (2/3) * 0.8 + (1/3) * 0.7 = 0.76$$

3 Evaluation of Requests

3.1 Principle

Here we propose algorithms which allow us to determine whether, for a set X resulting from a partitioning, the value of truth of the predicate "Q x's are A" is greater than a given threshold α (degree of satisfaction). We will assume that: i) the number n of elements of each set is known (this could be easily added in a usual index structure or implemented in a sort algorithm), ii) each set can be accessed separately but step by step as tuples are required. Our aim is to replace algorithms in $\Theta(n)$ (based on naïve strategies) by algorithms in $O(n)$ (based on improved strategies). Therefore, the objective is to reduce the access to tuples and by doing this, to indicate those conditions deciding whether the calculus should continue or can stop. More precisely, the calculus (mainly the loop wich encompasses data access) can stop in two circumstances: i) when the set cannot reach the desired level (α), ii) when it is certain that the set will reach the desired level (α) and the precise value of the membership degree is not required. This reasoning is very similar to what is done in the design of "try and error" or "branch and bound" algorithms where some heuristics is searched in order to limit the search tree, i. e. the number of candidates to be examined.

3.2 Evaluation of Relative Quantifiers According to Zadeh

According to Zadeh, the evaluation of the set-oriented predicate "Q_r x's are A" (Q_r being a relative quantifier) for a set $X=\{x_1, ..., x_n\}$ is $Q_r(\sum_X \mu_A(x)/n)$. By giving ourselves a threshold α, we revert to determining the sets X satisfying $Q_r(\sum_X \mu_A(x)/n) \geq \alpha$. According to the curve (cf fig. 3) representing the quantifier Q_r, we can consider that our problem is turned into the characterization of the sets $X=\{x_1, ..., x_n\}$ satisfying $a \leq \sum_X \mu_A(x)/n \leq b$ (1)

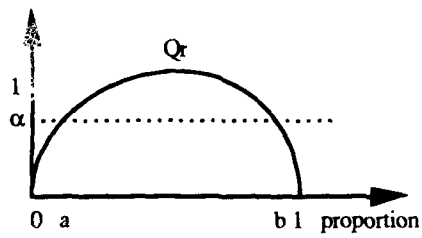


Fig. 3. An α-cut

Naïve Algorithm. The naïve algorithm consists in carrying out an exhaustive scan of the set $X=\{x_1, ..., x_n\}$. Let us write $S_k=\sum_{i=1}^{k}(\mu_A(x_i)/n)$. We are seeking to obtain and then test the value S_n.

S_0:=0;
for i:=1 *to* n *do*
  S_i:=S_{i-1}+$\mu_A(x_i)$/n;
endfor;
if $S_n \in [a,b]$
then the set satisfies the predicate with a degree greater than (or equal to) α
else the set satisfies the predicate with a degree smaller than α;
endif;

Improved Algorithm. The idea is to modify the naïve algorithm using heuristics which allow us to predict whether or not the set will confirm the double inequation (1). Let $k \in [0, n-1]$, we have:

$$S_n - S_k = \sum_{i=k+1}^{n} (\mu_A(x_i)/n).$$

Now $\forall i, \mu_A(x_i) \in [0,1]$, wich implies that, $0 \leq S_n - S_k \leq (n-k)/n$. Thus we can write:

$$\forall k \in [0,n-1]\ S_k \leq S_n \leq S_k + (n-k)/n.$$

Therefore, we can conclude that there are two heuristics to stop the algorithm at step k of the iteration:

If $S_k \in [a,b]$ and if $S_k + (n-k)/n \in [a,b]$ then it is certain that $S_n \in [a,b]$.Therefore, we can stop the iteration because the set satisfies the predicate with a degree greater than or equal to the threshold (it is a success).

If $S_k > b$ or $S_k + (n-k)/n < a$ then it is certain that $S_n \notin [a,b]$. Therefore, we can stop the iteration because the set satisfies the predicate with a degree smaller than the threshold (it is a failure).The improved algorithm, therefore, is :

S_0 := 0;
for k := 1 *to* n *do*
  $S_k := S_{k-1} + \mu_A(x_k)/n$;
  if $S_k > b$ *or* $S_k + (n-k)/n < a$ *then exit failure endif*;
  if $S_k \in [a,b]$ *and* $S_k + (n-k)/n \in [a,b]$ *then exit success endif*;
endfor;
exit success: the set satisfies the predicate with a degree greater than (or equal to) α.
exit failure: the set satisfies the predicate with a degree smaller than α.

3.3 OWA Based Evaluation of Quantifications

We present, for a considered set X, the OWA based evaluation of the predicate "Q x's are A", assuming that Q is a monotonic quantifier. First we present a naïve algorithm to compute the OWA aggregation. Then, we recall the properties of an OWA. These properties will be used to specify an improved algorithm from the naïve one.

Naïve Algorithm. The naïve algorithm performing the calculus of the OWA aggregation is based on the exhaustive scan of the set. It is assumed that W is the

vector that describe the weight of the OWA (the W[i] are computed from Q), the x_i's are the element of the set X and GV is the result of the aggregation.

for each x_i *in* X *do* V[i] = $\mu_A(x_i)$ *endfor;*
order the vector V giving V';
compute the vector W;
comment this calculus only needs the knowledge of n the cardinality of X *endcomment*;
GV = 0;
for i *from* 1 *to* n *do* GV = GV + V'[i] * W[i] *endfor;*
if GV ≥ α *then* the set X satisfies the quantification with a degree of GV greater than or equal to α *endif;*

Some Properties of the OWA Operator. The OWA is a mean operator and so it has interesting properties:

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \le OWA(w_1, \dots, w_n, x_1, \dots, x_i, 1, \dots, 1) \quad (2)$$
$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \le OWA(w_1, \dots, w_n, 1, 1, \dots x_i, 1, \dots, 1) \quad (3)$$
$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \ge OWA(w_1, \dots, w_n, x_1, \dots, x_i, 0, \dots, 0) \quad (4)$$

which arise from: i) the monotonicity of any mean operator, ii) the fact that x belongs to [0,1]. From these basic properties, one can derive conditions bearing on the x_i's which are necessary for the satisfaction of the condition:

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \ge \alpha.$$

From (3), one can derive:

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \ge \alpha$$

$$\Rightarrow OWA(w_1, \dots, w_n, 1, \dots, 1, x_i, 1, \dots, 1) \ge \alpha \Leftrightarrow \left(\sum_{i=1}^{n-1} w_i * 1\right) + w_n * x_i \ge \alpha$$
$$\Leftrightarrow (1 - w_n) + w_n * x_i \ge \alpha$$

and finally, we get:

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \ge \alpha \Rightarrow \forall i,\ x_i \ge \frac{\alpha + w_n - 1}{w_n} \quad (5).$$

This last formula is valid only if w_n is strictly positive, otherwise no implication can be found. Moreover, it is only profitable if $(\alpha + w_n) > 1$ (otherwise, we have a condition wich is trivially satisfied). From (2) and (4), we have:

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, 1, \dots, 1) < \alpha \Rightarrow OWA(w_1, \dots, w_n, x_1, \dots, x_n) < \alpha \quad (6)$$

$$OWA(w_1, \dots, w_n, x_1, \dots, x_i, 0, \dots, 0) \ge \alpha \Rightarrow OWA(w_1, \dots, w_n, x_1, \dots, x_i, \dots, x_n) \ge \alpha \quad (7)$$

These last two conditions will be used for partial evaluations of an OWA aggregation as we will see in the next section.

Improved Algorithm. Since n is known, the w_i can be calculated and in particular the last value w_n. Thus, if the sum $(\alpha + w_n)$ is greater than 1, we can apply the condition (5) to any tuple x_i of a set and insert the following instruction:

if $\mu_A(x_i) < \dfrac{\alpha + W[n] - 1}{W[n]}$ *then* exit failure *endif;*

From a practical point of view, sets with a large number of elements will lead to a low value for w_n and consequently this condition will not work frequently (unless if α is very close to 1). Now, let us assume that we have already accessed k tuples of a set (tuples x_1 to x_k), and the values $v_i = \mu_A(x_i)$ for $i \in [1, k]$ are known. If we assume that the (n-k) missing values are 1 and the result of the OWA aggregation remains smaller than α, according to formula (6) we can be certain that this set will never reach the desired level α. We have to determine the aggregation (we recall that v_{j_i} is the i^{th} largest value of the v_j's):

$$OWA(w_1, \ldots, w_n, v_1, \ldots, v_k, 1, \ldots, 1) = \sum_{i=1}^{n-k} w_i + \sum_{i=1}^{k} (w_{n-k+i} * v_{j_i})$$

This computation requires only that the values V[1] to V[k] are sorted. In addition, the expression does not have to be calculated from scratch from step k to step (k + 1) since, for the first part, it is enough to subtract w_{n-k}. Thus, once again, we can specify a condition likely to stop the outer loop:

insert $\mu_A(x_k)$ into V[i:k]; *comment* V[i:k] in decreasing order *endcomment;*
compute $A = \sum_{i=1}^{n-k} W[i] + \sum_{i=1}^{k} (W[n-k+i] * V[i])$;
if $A < \alpha$ *then* exit failure *endif;*

When k = n (last tuple of the set), the value of A equals precisely the value GV which is the degree tied to the set.

Finally, if the value of the membership degree of the set is not necessary, we can take advantage of formula (7). In fact, this formula states that if we have already accessed k tuples of a set (tuples x_1 to x_k) and we assume that the (n-k) missing values equal 0 and the result of the OWA aggregation already exceeds α, then we can be sure that the set will reach the desired level α. We have to determine:

$$OWA(w_1, \ldots, w_n, v_1, \ldots, v_k, 0, \ldots, 0) = \sum_{i=1}^{k} (w_i * v_{j_i})$$

Here again, we only have to sort values V[i] to V[k] and we can specify a condition likely to stop the outer loop:

insert $\mu_A(x_k)$ into V[i:k]; *comment* V[i:k] in decreasing order *endcomment;*

compute $B = \sum_{i=1}^{k}(W[i] * V[i])$;

if $B \geq \alpha$ *then* exit success *endif;*

Again, when k = n, the value of B equals that of A and GV is the membership degree of the current set. We can now give the final algorithm, when n the number of tuples of any set is known in advance:

compute the vector W; *comment* $W[i] = w_i$ *endcomment;*
for each x_k *in* X *do*
  if $\mu_A(x_k) < \frac{\alpha + W[n] - 1}{W[n]}$ *then* exit failure *endif;*
  insert $\mu_A(x_k)$ into V[i:k]; *comment* V[i:k] in decreasing order *endcomment;*
  $A = \sum_{i=1}^{n-k} W[i] + \sum_{i=1}^{k}(W[n-k+i] * V[i])$;
  if $A < \alpha$ *then* exit failure *endif;*
  $B = \sum_{i=1}^{k}(W[i] * V[i])$;
  if $B \geq \alpha$ *then* exit success *endif;*
enddo;
exit success: the set X satisfies the quantification with a degree greater than (or equal to) α.
exit failure: the quantification will never reach the threshold α.

3.4 An Example

Let us consider the query : "find the departments where most of the employees are well-paid, is satisfied with a degree greater than 0.73" which is expressed in SQLf as: *select* 0.73 dep *from* EMPLOYEE *group by* dep *having* most *are* well-paid. We examine a department (set of tuples sharing the same value for the attribute department) containing five employees e1 to e5 with the following characteristics:

emp	dep	salary
e1	d	38000
e2	d	55000
e3	d	46000
e4	d	32000
e5	d	48000

Fig. 4. A department

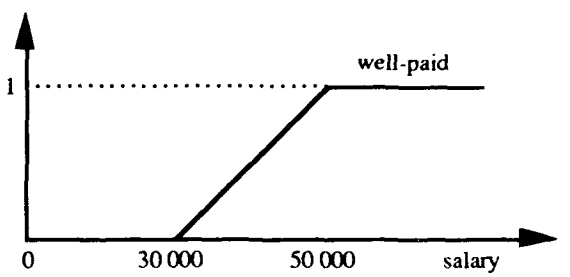


Fig. 5. A fuzzy predicate: well-paid

"Most" is represented by the function : $x \rightarrow x^2$, "well-paid" is the membership function given in figure 5. So the fuzzy set well-paid is {0.4/e1, 1/e2, 0.8/e3, 0.1/e4, 0.9/e5}.

Evaluation According to Zadeh's Interpretation of Quantifiers. "Most" is represented by the function : $x \rightarrow x^2$ thus a = 0.85 and b = 1. (because $x \in [0,1]$ and $x^2 \geq 0.73$ is equivalent to $x \in [0.85,1]$). If we perform the overall calculus presented by Zadeh we get $[(0.4 + 1 + 0.8 + 0.1 + 0.9)/5]^2 = 0.4$; therefore, this set does not match our requirement (0.73). Now, let us apply the improved algorithm presented in 3.2 assuming that the tuples are accessed according to the order depicted above:

Access to employee e1 : $\mu_{well\text{-}paid}(e1)=0.4 \Rightarrow S_1=0.08$, $S_1+(4/5) = 0.88 \Rightarrow$ the loop goes on
Access to employee e2 : $\mu_{well\text{-}paid}(e2)=1 \Rightarrow S_2=0.28$, $S_2+(3/5) = 0.88 \Rightarrow$ the loop goes on
Access to employee e3 : $\mu_{well\text{-}paid}(e3)=0.8 \Rightarrow S_3=0.44$, $S_3+(2/5) = 0.84 \Rightarrow$ *the loop stops because* $S_3 + 2/5 < 0.85$.

In this case we save 2 accesses. Moreover, if e4 were the first tuple of the considered set, the loop would have stopped immediately because (0.1/5 + 4/5) = 0.82 is less than 0.85.

Evaluation According to Yager's Interpretation of Quantifiers. The weight vector W is: W[1] = 0.04, W[2] = 0.12, W[3] = 0.2, W[4] = 0.28, W[5] = 0.36. If we perform the overall calculus for these data (naïve strategy requiring the access to the 5 tuples), we get: (0.04 * 1) + (0.12 * 0.9) + (0.2 * 0.8) + (0.28 * 0.4) + (0.36 * 0.1) = 0.45; therefore this set does not match our requirement (0.73). Now, let us apply the improved algorithm presented in 3.3 assuming that the tuples are accessed according to the order depicted above. Since (α + W[5]) is greater than 1, ((α + W[n] - 1)/W[n]) = 0.25), the first condition of the algorithm is interesting (not trivially satisfied).

Access to employee e1: $\mu_{well\text{-}paid}(e1) = 0.4 > 0.25$; A = 0.78 > 0.73; B = 0.01 < 0.73 $\Rightarrow$ the loop goes on
Access to employee e2: $\mu_{well\text{-}paid}(e2) = 1 > 0.25$; A = 0.78 > 0.73; B = 0.08 < 0.73 $\Rightarrow$ the loop goes on
Access to employee e3: $\mu_{well\text{-}paid}(e3) = 0.8 > 0.25$; A = 0.72 > 0.73 *is false* $\Rightarrow$ *the loop stops here*

In this case, we save also 2 accesses and if e4 were the first tuple of the considered set, the loop would also have stopped immediatly, since $\mu_{well\text{-}paid}(e4) = 0.1$ is under 0.25 and 4 data accesses would have been saved.

4 Conclusion

In this paper we have dealt with database management systems which support imprecise queries and in which conventional data are stored. More precisely, we have concentrated on a single type of fuzzy queries involving quantifiers and set-oriented

predicates. The evaluation of such a query leads to apply sequentially a predicate "Q elements of X are A" (assuming Q is a quantifier and A a fuzzy predicate) to several sets X resulting from a partitioning. We were interested in two interpretations of quantifiers, Yager's interpretation which can apply only to proportional (i. e. monotononic increasing) quantifiers and Zadeh's interpretation which was restricted to relative quantifiers. Both interpretations use a mean operator to accomplish their calculus.

We have designed some strategies for the evaluation of the considered quantified set-oriented fuzzy queries when a threshold for the degree of satisfaction is given by the user. Starting from a naïve strategy based on the exhaustive scan of a considered set, we have pointed out some properties of the quantifiers interpretations allowing for some improvements, especially regarding data access. Where the number of elements of the considered set is known, we have shown that conditions could apply to each element of the set to decide whether or not the calculus had to be continued. These conditions are based on the mean operators' property of monotonicity which give some heuristics to determine whether or not the set would reach the specified threshold. In the near future, we will perform some simulations in order to get an idea about the benefit given by our improvements.

References

1. P. Bosc, M. Galibourg, G. Hamon; Fuzzy querying with SQL: extension and implementation aspects. Fuzzy Sets and Systems 28, 333-349 (1988)

2. P. Bosc, O. Pivert; About equivalences in SQL^f, a relational language supporting imprecise querying. Proc. International Fuzzy Engineering Symposium, Yokohama (Japan): 309-320 (1991)

3. D. Dubois, H. Prade; A review of fuzzy set aggregation connectives. Information Sciences 36, 85-121 (1985)

4. H. Prade; A Two-Layer Fuzzy Pattern Matching Procedure for the Evaluation of Conditions Involving Vague Quantifiers. Journal of Intelligent and Robotic Systems 3, 93-101 (1990)

5. R.R. Yager; On ordered weighted averaging aggregation operators in multicriteria decisionmaking, IEEE Transactions on systems, Man and Cybernetics 18, 183-190 (1988)

6. R.R. Yager; Connectives and quantifiers in fuzzy sets, Fuzzy Sets and Systems 40, 39-75 (1991)

7. R.R. Yager; Fuzzy quotient operators for fuzzy relational databases, Proc. International Fuzzy Engineering Symposium, Yokohama (Japan): 289-296 (1991)

8. L.A. Zadeh; A computational approach to fuzzy quantifiers in natural languages, Computer Mathematics with Applications 9, 149-183 (1983)

Methodologies for Knowledge-Based Software Engineering

Michael R. Lowry

AI Research Branch, MS 269-2
NASA Ames Research Center / Recom Technologies
Moffett Field, CA 94035

Abstract. As the science of knowledge representation and automated reasoning advances, AI has the potential to radically change the artifacts, methodologies, and life cycles of software engineering. The most significant change will be when problems are formalized at the level of specifications rather than programs. This will greatly facilitate software reuse and modification. Achieving this potential requires overcoming many technical challenges, particularly the semi-automated synthesis of efficient and correct programs from specifications. The first part of this paper describes several methodologies for program synthesis and compares their ability to control the combinatorial explosion inherent in automated reasoning.

As knowledge-based software engineering matures and increasingly automates the software engineering life cycle, software engineering resources will shift toward knowledge acquisition and the automated reuse of expert knowledge for developing software artifacts. The second part of this paper describes methodologies for expanding the software life cycle to the knowledge life cycle.

1 Introduction

In the early sixties large software projects, such as those undertaken for NASA's Apollo program, forced software engineering to mature from an ad hoc endeavor practiced by small teams of programmers to a structured engineering discipline. Structured programming methodologies were developed to cope with the complexities of managing, specifying, designing, and implementing large software systems. Structured designs were captured through hand drawn diagrams depicting everything from project decomposition to data and control flow. CASE (computer-aided software engineering) emerged in the eighties when it became economically feasible to computerize structured programming by providing graphical user interfaces to manipulate these diagrams.

KBSE (knowledge-based software engineering) is a much more ambitious endeavor than current approaches to CASE. The key observation is that the current practice of modern software engineering lacks the sound mathematical basis characterizing other engineering disciplines. This limits the complexity of software systems that can be constructed with a high degree of reliability. Formal methods, the application of mathematical logic to software engineering, is just beginning to have an impact on real software engineering practice. The goal of KBSE is nothing less than the computerization of formal methods for all phases of the software life cycle [8].

KBSE addresses the essential tension between problem specification and efficient solution implementation. This tension makes it difficult to modify and reuse programs,

since efficient code incorporates constraints from all parts of a problem specification in the optimization of individual program fragments. Hence, local incremental changes to a problem specification often require extensive non-local changes to optimized code. Modification of production quality code is so time consuming that maintenance costs currently dominate software life cycle resources. Furthermore, reuse of production quality code has been difficult to achieve. Advances in programming language and compiler technology have raised the level of programming abstractions, but have not addressed the essential difference between optimized code suitable for efficient computation and formal problem specifications suitable for reuse and modification. The current paradigms for programming languages cannot in principle bridge this gap, because to guarantee compiler performance the peephole on the source code used in optimizing machine language code must be limited.

KBSE bridges this gap by introducing a new software development paradigm: problems are first formalized at the level of the declarative semantics of an application domain, and then semi-automatically transformed to the operational semantics of an efficiently compileable programming language [8]. Formal methods provide the mathematical basis for this transformation. Automated reasoning provides the means for carrying out the transformation. By raising the level at which problems are formalized, modification and reuse will be greatly facilitated. Furthermore, by introducing formal artifacts earlier in the software life cycle, mechanized support can be provided for the full spectrum of software engineering activities, from requirements engineering to validation and maintenance.

Evolutionary improvements of current software engineering methodologies can be achieved with existing KBSE technology. Particularly promising are domain-specific program synthesis tools and re-engineering tools to modify and maintain existing code. However, achieving the full KBSE paradigm will require many technical advances. Foremost are search control for automated reasoning, and interactive assistance in requirements formalization and validation

Raising the level at which problems are formalized from the programming level to the specification level will eliminate many conceptual and design errors. These errors can cost over a hundred times more to fix during the testing phase than simple coding errors [1]. This is one major motivation for applying formal methods even without computer-aided assistance. However, even at the specification level, formalization is a difficult process, and many of the most costly software errors can be traced back to the transformation from informal requirements to specifications. New methods for specification validation are needed. AI programming environments have already contributed to one method, rapid prototyping, in which executable specifications are developed in a very high level programming language and then validated interactively with end users. Other AI approaches to specification validation are described in [17, 18, 24] and their references.

This paper first describes several methodologies for program synthesis with an emphasis on search control, drawing upon the literature and the author's own work. It then contrasts knowledge-based methodologies for software engineering with methodologies based on CASE, and describes the evolution from the software engineering life cycle to the knowledge engineering life cycle.

2 Methodologies for Program Synthesis

2.1 Constructive Theorem Proving

Soon after Robinson [27] developed resolution as the first practical means for automated theorem proving in predicate logic, it was applied to automatic program synthesis. Green [6] and Waldinger [33] demonstrated the generation of small programs such as sorting algorithms through constructive proofs from specifications of the form:

$\forall x \exists y$ Precondition$(x) \rightarrow$ Postcondition(x, y)

In this specification schema x is a vector of input variables, y is a vector of output variables, and Precondition(x) is a formula constraining the input variables. A constructive proof binds y to a term which makes the specification a theorem. If this term is composed of functions in the programming language and the only variables in the term are input variables, then the term represents a functional program. The inference process is similar to logic programming: first the universally quantified variables in x are replaced by unique constants, the formula Precondition(x) is asserted over these constants, and then Postcondition(x,y) is negated and resolution is repeatedly applied until a refutation is derived. The program term is built up through unification with the output variables y. Recursive and iterative constructs are derived through inductive proofs using inductive schemata or through additional inference rules. Manna and Waldinger [19] later developed an elegant nonclausal variation suitable for manually controlled derivations.

Initially it was hoped that advances in generic theorem proving strategies would sufficiently control search to enable automated derivations to scale up to large problems. While impressive progress was made during the early seventies [3], generic resolution strategies were never able to mitigate combinatorial explosion effectively. Program synthesis often requires deep reasoning; generating recursive and iterative programming constructs through inductive proofs considerably expands the combinatorial explosion. In retrospect, it is unlikely that general purpose theorem-proving strategies will ever be sufficient to control the combinatorial search inherent in automated program synthesis.

2.2 Program Transformations

An alternative approach to program synthesis is incremental transformation of specifications to implementations through program transformations, i.e. oriented rewrite rules. In its purest form, the transformational approach is formalized by the semantics of conditional equational logic. In its more restricted variants the transformational approach can greatly reduce search [23]. It is also more adaptable than theorem proving to less formal knowledge engineering approaches, and can be viewed as an extension of current compiler technology. Transformations are typically oriented from higher level specification constructs to lower level implementation constructs, thus providing an overall direction to the search. Sets of rewrite rules are characterized by properties such as termination and confluence (guaranteed termination in a unique normal form). The Knuth-Bendix completion procedure [12], given an appropriate weighting scheme for terms and a set of rewrite rules, will add rewrite rules until the set becomes confluent. The Knuth-Bendix completion procedure is not guaranteed to terminate, and generally works only on small sets of rules and for restricted kinds of weighting schemes.

Despite the well-behaved search properties for restricted variants of program transformations, obtaining the same problem solving power as constructive theorem proving ultimately requires addressing the same combinatorial search issues. One way of increasing problem solving capability is to make the conditions for applying an inference rule more complex; but as the complexity is increased the amount of inference required typically increases exponentially. Furthermore, introducing general capabilities for producing recursive and iterative constructs requires expanded capabilities such as folding [4], in which a rewrite rule is reversed to introduce the application of a function from an instance of its definition. This reversal of rewrite rule orientation leads to a combinatorial explosion of possibilities. In general, as the scope of a transformation system is expanded to encompass a larger set of possible programs as output, the search space expands drastically.

2.3 *Manually Guided Program Synthesis/Verification*

At the opposite extreme to totally automated search control, several early systems (e.g.[2]) had the user select each primitive step of the inference or transformation process. Initially it was hoped that this approach would be a viable means for mechanically assisted program synthesis or verification. However, the sheer number of steps required made this approach infeasible outside of research settings or in applications such as avionics requiring extreme reliability. It was far easier to develop a program by hand than guide an inference system through the large number of primitive steps.

The problems with totally automated search control using generic strategies and the other extreme of manual guidance of primitive inference/transformation rules has led to methodologies centered on human/computer partnerships and reuse. These include the intelligent assistant approach in which humans make strategic decisions while the computer carries out bounded searches, reuse of generic programs or derivations, and encoding of tactical and strategic program design knowledge. Each of these methodologies introduces interacting knowledge representation and automated reasoning issues.

2.4 Intelligent Programming Assistant

Floyd [5] presented an early vision of an intelligent programming assistant, in which the computer kept track of clerical details while the human made the important strategic decisions. A key issue is developing representations for program derivations that are human comprehensible and machine manipulable. The decision making also has to be factored to limit the search carried out by automated reasoning while presenting meaningful strategic decisions for the human user. These constraints rule out certain technologies such as clausal resolution.

The programmer's assistant project at MIT, spanning the years 1973 to 1992, was an influential effort particularly in the area of re-engineering. The main achievements in the early years were the development of the plan formalism, a language independent representation for programs and programming knowledge, and demonstration of KBEmacs, an editor for manipulating programs in this formalism.

The plan formalism represented programs as flowcharts with explicit data and control flow arcs. The main innovation of the plan formalism was support for programming clichés, which are reusable algorithmic fragments (such as enumeration over a file) that were engineered to correspond to expert human programming knowledge. Analyzers were developed for several programming languages that recognized instances of clichés

in program text. The combination of analyzers and translators between the plan formalism and program text enabled significant re-engineering capabilities, such as modification of programs at the level of programming clichés and improved translation of programs between programming languages via abstraction to the plan formalism and then reimplementation in the target programming language [34].

However, the plan formalism lacked the semantic basis to provide generality and power; reasoning was carried out by ad hoc procedures. This limited the feasibility of extending the capabilities of KBEemacs. To address these limitations Rich [25] formalized the plan formalism into the plan calculus and then developed CAKE [26], a layered automated reasoning system. CAKE is a careful integration of different automated reasoning capabilities (e.g. truth maintenance, propositional reasoning, equality, and types) that appears as an active knowledge base for software artifacts. CAKE's automatically invoked inference procedures are constrained to run in polynomial time; user queries can invoke more time consuming reasoning procedures. Significant new capabilities for the programmer's apprentice were implemented on top of the plan calculus and CAKE, including a debugging assistant [14] and a requirements assistant (RA) [28]. The RA is a good example of the interactive problem formalization assistance that can be provided through KBSE: the RA notified a user when it detected ambiguity, contradiction, incompleteness, or inaccuracy in an evolving requirements specification.

Several lessons can be learned from the evolution of MIT's programmer assistant project. First, although a knowledge engineering approach is useful in the initial development of a representation meaningful to humans, achieving generality and power requires a semantically well defined formal representation with semantically well founded inference procedures. Without these, the implicit assumptions which facilitate an ad hoc approach become limitations hindering the expansion of a KBSE system. These factors will limit the expansion of current CASE systems, because of their shallow representations and ad hoc reasoning procedures. Second, developing the automated reasoning capabilities to support a formal representation requires significant engineering to avoid combinatorial explosion.

2.5 Replaying Program Derivations

An alternative to reusing high-level generic program fragments such as clichés is the reuse of program derivations. This approach spans the range from rote replay of derivations to derivational analogy [20]. Program derivation reuse is particularly appealing because it has the potential to support the incremental modification of specifications by rederiving efficient implementations through replay of the original derivation. When the replay system encounters part of the derivation which is no longer applicable, then it transfers control to the user.

Derivational analogy replay systems have been successfully applied in domains such as VLSI design where the mapping from the input of the derivation system to the output is localized; that is, each part of the output is attributable to localized parts of the input. However, as discussed earlier, optimized code must potentially incorporate constraints from all parts of a specification. For this reason, substantial parts of the original program derivation might no longer be applicable after an incremental change in specification. To date, most derivational analogy replay systems for program synthesis have operated on representations of the enablement structure of transformation or inference rules. There is typically no representation in the derivation record of the purpose for applying a transformation in meeting a performance goal for the optimized code (however, see [35]).

Thus there is insufficient information for making good analogies to other derivations when parts of the original derivation are no longer applicable. For these reasons, derivational analogy replay systems have had little more success so far than rote replay systems in program synthesis.

2.6 Design Analysis

While generic theorem proving or transformational strategies have had limited success in automating program synthesis, a more promising methodology is to develop efficient tactics for controlling automated reasoning for particular classes of software artifacts. Each tactic can be viewed in itself as a special purpose program synthesizer. However, because tactics are control programs for general purpose inference mechanisms, they can be easily combined. The following describes one methodology for developing tactics within the context of parameterized theories and algebraic specifications. The example used is the development of a tactic for synthesizing local search algorithms, more details can be found in [15].

Design analysis is a methodology for formalizing both the structural properties common to a class of software artifacts and the genetic properties common to their derivations [30]. This formalization is then used to develop a design tactic that automatically designs an artifact in this class given a specification of its behavior. Design analysis formalizes intrinsic structural properties rather than properties specific to a particular programming language or application domain. By abstracting away these particular concerns, the resulting formalization is more broadly applicable. The objective is to find a general mathematical characterization of the structure of a class while at the same time capturing the features that provide search guidance for designing artifacts in a class.

The first step of design analysis for algorithm synthesis is to study many examples of a naturally defined class of algorithms. For example, local search algorithms, also referred to as hill–climbing algorithms, are a natural class in which a feasible solution to an optimization problem is iteratively improved by searching a neighborhood of the solution for a better solution, and stopping when no neighboring solution is better. The second step of design analysis is to extract the features and structural constraints characterizing that class of algorithms. The neighborhood structure determines the properties of a local search algorithm : exact neighborhood structures guarantee that local optimums are global optimums, while the weaker condition of reachability guarantees that all feasible solutions for a given input are mutually reachable. Reachability is a necessary condition for variants of local search that can bactrack out of local optimums, such as simulated annealing, to converge on global optimums. The third step is to formalize this characterization in a theory. The theory of neighborhood structures for local search algorithms is an extension of the theory for optimization problems.

A basic problem is specified by defining a set of inputs **D**, a set of outputs **R**, an operation **I** that maps legal inputs to true, and an operation **O** that maps input/output pairs to true when the output is a feasible solution to the input. A basic problem specification is a tuple **B= <D,R,I,O>**.

An optimization problem is specified by extending a basic problem specification with an ordering relation in which all pairs of feasible solutions are comparable. All such ordering relations can be formulated as a cost function that maps feasible solutions to a totally ordered set. For most problems the cost function maps feasible solutions to the integers, rationals, or reals. The totally ordered set is denoted $\langle \Re, \leq \rangle$, where $\Re$ is the set

and $\leq$ is the total order relation. Thus an optimization problem is specified through a tuple $\mathbf{Opt}=\langle \mathbf{D},\mathbf{R},\mathbf{I},\mathbf{O},\Re,\leq,\text{cost}\rangle$.

A local search theory $\mathbf{LS} = \langle \mathbf{Opt},\mathbf{N}\rangle$ is specified by an optimization problem and a neighborhood relation. Three axioms, two being optional, constrain the neighborhood relation, which is a ternary relation between an input and two elements of the output domain. First, each feasible solution is in its own neighborhood, so that for any legal input the neighborhood relation is a reflexive relation on feasible outputs (Axiom LS1). If the neighborhood structure is exact, then the local search theory will be called exact (Axiom LS2). Likewise, if the neighborhood structure is reachable, the local search theory will be called reachable (Axiom LS3). A local search theory for a particular optimization problem is defined by a mapping from the components of abstract local search theory to definitions of objects, functions and relations in the problem domain. More formally, the mapping is a theory interpretation, which means that the abstract axioms are true when they are mapped to the problem domain theory. Abstract local search theory is defined as follows:

Sorts $\mathbf{D},\mathbf{R},\Re$

Operations

$\mathbf{I}:\mathbf{D}\rightarrow$ boolean

$\mathbf{O}:\mathbf{D}\times\mathbf{R}\rightarrow$ boolean

$\mathbf{cost}:\mathbf{D}\times\mathbf{R}\rightarrow\Re$

$\leq:\Re\times\Re\rightarrow$ boolean

$\mathbf{N}:\mathbf{D}\times\mathbf{R}\times\mathbf{R}\rightarrow$ boolean

$\text{Optimal}(x,y)\equiv\forall(y')\mathbf{O}(x,y')\Rightarrow\mathbf{cost}(x,y)\leq\mathbf{cost}(x,y')$

Axioms

LS1: **Reflexive Neighborhood**

$$\forall(x,y)\ \mathbf{I}(x)\wedge\mathbf{O}(x,y)\Rightarrow\mathbf{N}(x,y,y)$$

LS2: **Exact Neighborhood**

$$\forall(x,y)\ \mathbf{I}(x)\wedge\mathbf{O}(x,y)\wedge\left[\forall(y')\ \mathbf{O}(x,y')\wedge\mathbf{N}(x,y,y')\Rightarrow\mathbf{cost}(x,y)\leq\mathbf{cost}(x,y')\right]\Rightarrow\text{Optimal}(x,y)$$

LS3: **Reachable Neighborhood**

$$\forall(x,y,y')\ \mathbf{I}(x)\wedge\mathbf{O}(x,y)\wedge\mathbf{O}(x,y')\Rightarrow\mathbf{N}^{*}(x,y,y')$$

where $\mathbf{N}^{*}$ is the reflexive and transitive closure of $\mathbf{N}$:

$$\forall(x,y)\mathbf{N}^{0}(x,y,y)\equiv\mathbf{I}(x)\wedge\mathbf{O}(x,y)$$

$$\forall(k\in Nat;x,y,y')\mathbf{N}^{k+1}(x,y,y')\equiv\exists(z)\mathbf{O}(x,z)\wedge\mathbf{N}^{k}(x,y,z)\wedge\mathbf{N}(x,z,y')$$

$$\forall(x,y,y')\mathbf{N}^{*}(x,y,y')\equiv\exists(k\in Nat)\mathbf{N}^{k}(x,y,y')$$

To derive a local search algorithm for a particular optimization problem, a partial mapping from this abstract local search theory to the components of an optimization problem is first created. Constraints for a suitable neighborhood relation are then derived by instantiating the abstract neighborhood axioms with these components. The main part of the design tactic is to derive the definition of a neighborhood relation from these constraints in terms of the problem domain. Once the neighborhood relation is defined,

an initial algorithm can be derived by instantiating a program schema with the components of the derived local search theory. This high-level algorithm can be further refined with optimization tactics such as partial deduction [13] and finite differencing [21].

Formalizing the structure of a class of software artifacts is by itself usually insufficient for providing mechanized design assistance; it is also necessary to formalize the structure of the derivations. In the example of local search algorithms, the axioms for reachability and exactness defined above are too general to avoid combinatorial explosion in automated reasoning. The axioms for reachability require induction over the transitive closure of neighborhoods, which can be difficult for automated theorem provers. The exact neighborhood axiom, as stated, does not provide sufficient structure for determing its satisfaction for most problems. (Typically, proofs for exact neighborhoods are done through reduction to problems with known exact neighborhoods such as linear programming, or through lemmas about convex functions.)

To provide heuristic adequacy for guiding derivations, various specializations of the general structure are derived. For local search algorithms whose neighborhoods are reachable but not necessarily exact, most neighborhoods for efficient local search algorithms can be described as natural perturbations of data structures: [The key step in deriving a local search algorithm is the] "... selection of a neighborhood or a class of neighborhoods, and this is tied to the notion of a 'natural' perturbation of a feasible solution" ([22] pg. 469). The theory of groups and group actions provides the mathematical basis for formalizing natural perturbations.

A natural perturbation neighborhood is defined for a data structure by a set of permutations and a group action mapping these permutations to perturbations of each instance of the data structure. Thus the neighborhoods for all instances of the data structure are similar extentionally and have the same intentional description based on the set of permutations. A permutation is any one-to-one (and hence invertible) function from some set of objects to the same set. The closure of this set of permutations under composition together with the group action defines three interrelated structures: a group of permutations, the mutually reachable data structures, and the invariant properties of mutually reachable data structures.

The specialization of reachable neighborhoods to natural perturbations entails only two restrictions on the reachable neighborhoods axiomatized in the abstract theory of local search. First, neighborhoods are required to be symmetric, that is if y is in x's neighborhood then x is in y's neighborhood. Most local search algorithms satisfy this condition. This condition ensures that if z is reachable from w then w is reachable from z. Second, the neighborhoods of all feasible solutions are similar; they have the same intensional description in terms of the set of permutations. These two restrictions are sufficient to enable the tools of group theory to be used in developing reachable neighborhood structures for a wide variety of optimization problems. The mathematics and proofs are fully developed in [15].

Specializing reachable neighborhoods to natural perturbation neighborhoods considerably simplifies automated reasoning. In particular, it is no longer necessary to do an inductive proof on the transitive closure of neighborhoods: reachability is ensured if and only if the invariant properties of a natural perturbation neighborhood are equivalent to the feasibility constraints for problem solutions. If the invariants are stronger than the feasibility constraints then some feasible solutions would not be reachable from other feasible solutions. If the invariants are weaker than the feasibility constraints then some feasible solutions would be mapped to infeasible solutions.

The local search design tactic developed in this approach matches a problem specification to a library theory whose invariants are equivalent or weaker than the feasibility constraints, and then specializes the library theory if the invariants are weaker. The library theories are defined for general set theoretic data structures, such as ordered sequences, as explained below. Reasoning about invariant properties and feasibility constraints provides a computationally tractable method of matching and then specializing theories in a library to problem specifications. The theorem prover does not have to reason directly about the second order reachability axioms - this has already been done by the creator of the library theories.

Library theories are based on basic neighborhoods which are the subclass of natural perturbation neighborhoods in which the permutations are restricted to be all the transpositions of some underlying set, that is, permutations in which only two elements are interchanged. For example, one basic neighborhood structure for an ordered sequence is defined by all the transpositions of the indices of the sequence. Basic neighborhoods are typically overly general for any particular problem; the design tactic first matches a problem specification to a basic neighborhood and then specializes the basic neighborhood. The current library of parameterized natural perturbation neighborhood theories consists of a half dozen basic neighborhood definitions, which include specifications of their invariants. A basic neighborhood has the following definition schema as a ternary relation, where y,y' are neighboring feasible solutions with respect to input x, and i,j are the elements that are transposed:

$\lambda\, x, y, y'.\exists(i, j \in S)\ y' = \mathbf{Action}(x, y, i, j)$

A local search library theory for a basic neighborhood consists of the basic neighborhood definition and definitions for the other components of a local search theory. It is presented as a mapping of the following form from abstract local search theory to a set of definitions:

LS - basic theory
$\mathbf{D} \mapsto \mathit{datatype1}(\alpha)$
$\mathbf{R} \mapsto \mathit{datatype2}(\alpha)$
$\mathbf{I} \mapsto \lambda x.\mathbf{P}(x)$
$\mathbf{O} \mapsto \lambda x, y.\ \mathbf{Invariant}(x, y)$
$\mathbf{N} \mapsto \lambda x, y, y'.\ \exists(i, j \in \mathbf{F}(x))\ y' = \mathbf{Action}(x, y, i, j)$

For example, the following mapping from abstract local search theory defines the basic neighborhood structure for same-sized subsets of a given finite set S. The size of the subsets are a constant size m, the elements which are transposed are the elements of the finite set:

LS - subset theory
$\mathbf{D} \mapsto \mathit{set}(\alpha) \times \mathit{integer}$
$\mathbf{R} \mapsto \mathit{set}(\alpha)$
$\mathbf{I} \mapsto \lambda S, m.\ m \leq \mathrm{size}(S)$
$\mathbf{O} \mapsto \lambda S, m, y.\ y \subseteq S \wedge \mathrm{size}(y) = m$
$\mathbf{N} \mapsto \lambda S, m, y, y'.\ \exists(i, j) i \in (S - y) \wedge\ j \in y \wedge\ y' = (y \cup \{i\}) - \{j\}$

This theory can be matched to a wide range of problems including the class of ACS (additive cost subset) problems defined by Savage. A typical example is the minimal

spanning tree problem (MST), which is to find a minimally weighted subset of edges in a graph that span the nodes of the graph without any cycles.

Given the specification of an optimization problem, the local search design tactic first matches the problem specification to a library theory for a basic neighborhood, and then specializes the library theory by finding necessary conditions on transpositions to ensure that feasible solutions are transformed to better feasible solutions. The design tactic takes the following steps; each step is a well defined inference problem with a manageable search space; the overall effect is to replace a large search space with a sequence of smaller search spaces:

1. Retrieve and match basic neighborhood theories from the library indexed by the type of feasible solution. A theory matches a problem specification if the invariants of the theory are necessary conditions of the feasibility constraints of the specification.
2. Determine necessary preconditions on the transpositions that ensure that a feasible solution is perturbed to a feasible solution.
3. Determine necessary preconditions on the transpositions that ensure that a feasible solution is perturbed to a better feasible solution.
4. (Optional step for deriving exact local search algorithms) Determine necessary conditions for a local optimum to be a global optimum.
5. Instantiate a program schema with the components of the theory where the derived preconditions on transpositions are guards on the application of a transposition.

Proofs for the correctness of the tactic and a detailed description illustrated with the derivation of the simplex algorithm can be found in [15]. This design tactic was developed as an extension of the KIDS system [31]. A generalization of the methods for matching library theories to problem specifications is presented in [32].

2.7 Summary of Methodologies for Program Synthesis

The previous subsections have reviewed various methodologies for automated or semi-automated program synthesis. In order to generate production quality code from high level problem specifications, a program synthesis system should be able to incorporate constraints from all parts of a specification into each program fragment. This cannot be achieved by translation systems which restrict the window on the source code considered in optimizing the output code.

The general dilemma is that the number of possible programs that can be generated from a given specification is combinatorially explosive. General purpose methods have not been found for efficiently searching this space for efficient programs; it is unlikely any such universal method exists. However, as described above, currently there are some effective techniques for capturing the knowledge used by expert human programmers and applying it in automated fashion. Even setting aside the issue of program correctness, it is worthwhile capturing this knowledge within a formal representation with semantically well-founded inference methods. This avoids the limitations that arise when trying to generalize and expand ad hoc methods. To avoid intractable searches during automated program synthesis, it is necessary for knowledge representations and inference methods to be carefully designed in tandem to limit search spaces. It is likely that as continued progress is made in automated program synthesis, improvements in formal knowledge representations will be driven as much by considerations of making inference tractable as by theoretical considerations in mathematical logic.

3 From the Software Life Cycle to the Knowledge Life Cycle

The previous section described various methodologies for program synthesis. While automated program synthesis is necessary to raise software engineering from the programming level to the specification level, it is only one component of the KBSE paradigm. As KBSE matures and increasingly automates the software engineering life cycle, software engineering resources will increasingly shift toward knowledge acquisition and the automated reuse of knowledge for developing software artifacts. This section describes how the various components of KBSE could interact.

Knowledge, like software, has its own characteristic life cycle. The knowledge life cycle is the maturation of design knowledge for an application domain from the initial research stage to the cookbook engineering stage. Knowledge-based design tools can provide support at stages of the knowledge life cycle that are not well supported with conventional software design tools. Furthermore, knowledge-based design tools have the potential of significantly compressing the knowledge life cycle.

One principle objective of KBSE is to compress the software life cycle with knowledge-based tools. By its very nature, knowledge-based design depends on the maturity of knowledge about an application domain. As knowledge about an application domain is developed by scientists and engineers, the design process for that domain makes a transition from creative and innovative design to routine and cookbook design. Different kinds of design tools are appropriate at different stages of the knowledge life cycle. One of the main anticipated advantages of KBSE is the ability to leverage the expertise of scientists and engineers by capturing their design knowledge at all stages of the knowledge life cycle.

An example of the knowledge life cycle is the development of the theory and technology for designing parsers used in compilers and other language-processing systems. Early parsers in the late fifties were ad hoc systems. Not only was there a lack of a theory of parsing to guide their design, there was not even a theory of grammars that specified the function of a parser. Thus the design of early parsers was creative: both the structure and function were unknown and ill-defined. The development of the BNF formalism in the early sixties clarified the function of a parser: to produce a trace of the BNF rules used to generate a text string from the text string itself. At this stage the design of parsers became innovative: the function was known, but the structure of possible solutions was still unexplored.

The mid sixties to the mid seventies witnessed rapid development of the theory and technology for parsing. First, recursive descent parsing was formalized, enabling parser development to become a routine design task. While routine, the design of these early recursive descent parsers required the configuration of a set of procedures, i.e., configurational design. By the late sixties, several table-driven parsers were developed: operator precedence, LL parsing, and LR parsing. (These early parsing formalisms did not handle left recursion, which required the development of LALR parsing.) Designing a table-driven parser is now a routine parametric design process, that is, the output of a design is a set of parameters for a specialized representation. When knowledge about an application domain becomes this advanced, then automated design tools can be readily developed with conventional software technology. Hence in the late sixties and seventies, parser generators were developed that take a specification of a grammar and automatically generate the parameters for a table-driven parser.

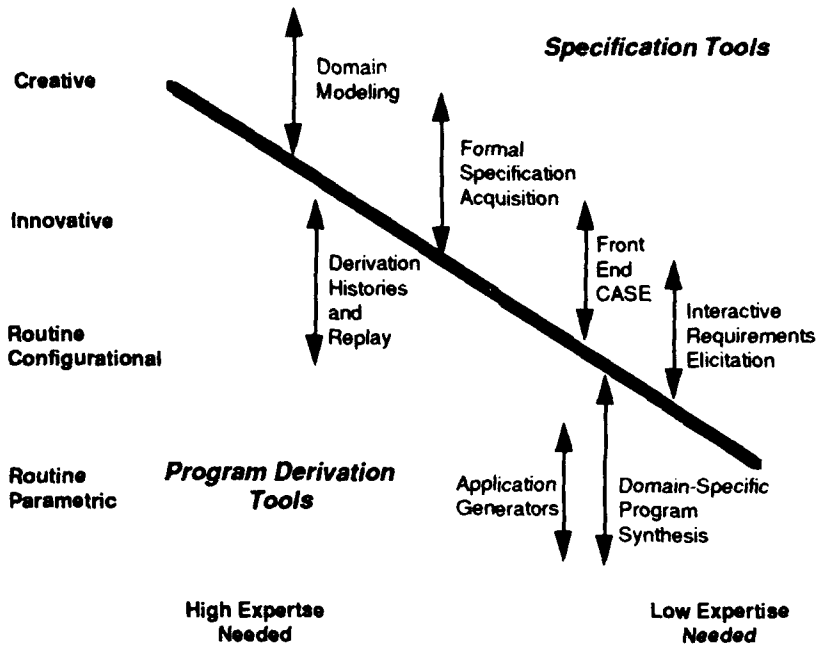


Figure 1. Spectrum of knowledge-based tools in the knowledge life cycle.

Figure 1 shows how knowledge-based design tools fit into the spectrum from creative design to routine parametric design. The horizontal axis denotes the level of human expertise required to use the tool, while the diagonal line separates specification tools from program derivation tools. The figure illustrates that knowledge-based tools can be used much earlier in the knowledge life cycle than current CASE tools. Domain modeling tools use a formal modeling language to express knowledge about an application domain. This knowledge can be used for different operational goals throughout the software life cycle, from requirements engineering to re-engineering. Domain modeling is the first step in moving from creative design to routine design. Because domain modeling is essentially the formalization of domain knowledge it requires a high degree of expertise, both in the application domain and in knowledge representation formalisms. During the innovative phase of the knowledge life cycle, general purpose interactive program synthesis systems could be used to explore the solution space for an application domain. Many of these general purpose systems have facilities for recording, editing, and replaying derivation histories. These replay facilities might enable users with less expertise than the original designer to develop derivations for similar specifications [20]. Given domain knowledge from a domain modeling tool, formal specifications can be developed and incrementally modified with tools such as ARIES [9].

Figure 1 also illustrates that knowledge-based tools can be employed by users with lower levels of expertise than required for current CASE tools. Front end CASE tools enable designers to define and edit software abstractions like data flow diagrams during

the initial stages of system design. However, because these CASE tools lack application domain knowledge they require more expertise and provide lower levels of verification and simulation capabilities than can be provided with knowledge-based tools. A good example of a knowledge-based specification tool is the WATSON system [11], which interactively elicits and validates requirements for new telephone features (such as call waiting) from telephony engineers using domain level scenarios. Back end CASE tools such as application generators (e g., parser generators) are currently widely used for the final stages of coding, particularly in commercial data processing. They consist of a menu driven or application language front end and a template-driven code generator back end. As such they are suitable for routine parametric design. In contrast, domain-specific program synthesis systems also can be used for routine configurational design and have a high level user interface that reduces the expertise needed by an end-user. Because systems like ELF [29] and SYNAPSE [10] combine template driven code generation with more powerful AI semantic processing techniques and transformation rules, they can tackle routine configurational design in addition to parametric design. They also produce more optimal code than an application generator.

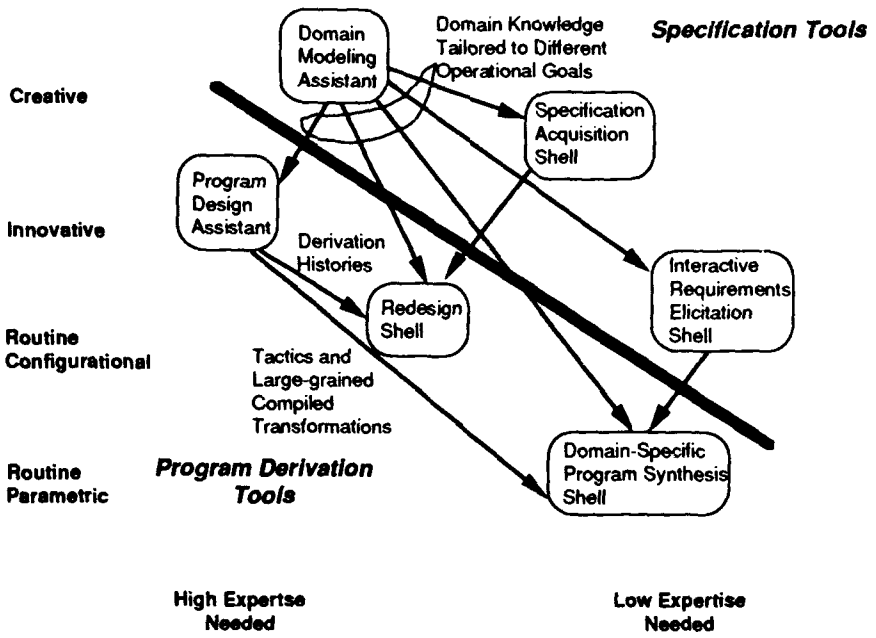


Figure 2. Transfer and reuse of expertise in the KBSE paradigm.

Knowledge-based technology, by providing an active medium for communication of knowledge, can also potentially compress the knowledge life cycle. In the absence of major advances in machine discovery, the development of design knowledge will continue to be a human-intensive process requiring high levels of expertise. However, knowledge-based tools could assist human research scientists and engineers in the development of this knowledge. These tools could then compile and transfer this

knowledge *into shells that* do interactive requirements elicitation, redesign, specification acquisition, and domain-specific program synthesis.

Figure 2 illustrates this transfer of expertise using knowledge-based subsystems; a more detailed exposition can be found in [16]. The domain modeling assistant and program design assistant would be used by scientists and engineers during the creative and innovative stages of the knowledge life cycle. The specification acquisition shell would support future system analysts in developing formal system specifications, while the redesign shell would enable system developers to construct software systems rapidly by editing and replaying derivation histories developed with a program design assistant.

For end-users with low expertise, requirements elicitation shells would use domain knowledge to develop interactively formal specifications of their requirements using informal examples. Domain-specific program synthesis shells would then synthesize a system meeting these requirements. Note that while a system developer with intermediate expertise could be expected to understand and manipulate derivation histories to develop a software system, an end user with low expertise would require totally automatic program synthesis. Thus a domain specific synthesis shell would require more highly compiled control knowledge for controlling software derivation than a redesign shell, i.e., tactics or large-grained compiled transformations as opposed to interpreting and manipulating derivation histories.

4 Conclusion

Knowledge based software engineering is based on research spanning over two decades. Significant commercial applications are likely within this next decade, particularly as industrial pilot projects in domain specific program synthesis and re-engineering mature. Interest in formal methods will also spur development of the field. However, to achieve the full paradigm requires many technical advances in knowledge representation and automated reasoning. This paper has described various methodologies for making automated reasoning tractable for program synthesis. Further improvements are likely to require that knowledge representations for program design expertise be developed in tandem with automated reasoning methods.

The paradigm of the knowledge life cycle can help to clarify the role of knowledge-based software engineering tools and guide their development. Different kinds of knowledge-based tools are appropriate at different stages of the knowledge life cycle. Furthermore, knowledge-based tools can expedite the transfer of expertise from research scientists and engineers and thus compress the knowledge life cycle.

References

1. B.W. Boehm: Software Engineering Economics. Englewood: Prentice Hall1981

2. M. Broy, P. Pepper: Program Development as a Formal Activity. IEEE Trans. on Software Eng. 7(1), 14-22 (1981)

3. C.L. Chang, R.C.T. Lee: Symbolic Logic and Mechanical Theorem Proving. New York: Academic Press 1973

4. J. Darlington: An Experimental Program Transformation and Synthesis System. Artificial Intelligence 16, 1-46 (1981)

5. R. Floyd: Toward Interactive Design of Correct Programs. In C. Rich, R.C. Waters (eds.): Artificial Intelligence and Software Engineering. Los Altos: Morgan Kaufmann 1986, pp. 331-334

6. C. Green: Application of Theorem Proving to Problem Solving. IJCAI (1969)

8. C. Green, D. Luckham, R. Balzer, T. Cheatham, C. Rich: Report on a Knowledge-Based Software Assistant. In C. Rich, R.C. Waters (eds.): Artificial Intelligence and Software Engineering. Los Altos: Morgan Kaufmann 1986, pp. 337-428

9. W.L. Johnson, M.S. Feather: Using Evolution Transformations to Construct Specifications. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 65-92

10. E. Kant, F. Daube, W. MacGregor, J. Wald: Scientific Programming by Automated Synthesis. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 169-206

11. V.E. Kelly, U. Nonnenmann: Reducing the Complexity of Formal Specification Acquisition. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 41-64

12. D.E. Knuth, P.B. Bendix: Simple Word Problems in Universal Algebras. In J. Leach (ed.): Computational Problems in Abstract Algebra. Pergammon Press 1970, pp. 263-298

13. J. Komorowski: Synthesis of Programs in the Partial Deduction Framework. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 377-404

14. R.I. Kuper: Dependency-directed localization of software bugs. Tech. Report 1053 MIT AI Lab, 1989

15. M.R. Lowry: Automating the Design of Local Search Algorithms. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 515-546

16. M.R. Lowry: Software Engineering in the Twenty-First Century. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 627-654

17. M.R. Lowry, R. Duran: Knowledge-based Software Engineering. In A. Barr, P.R. Cohen, E.A. Feigenbaum (eds.): The Handbook of Artificial Intelligence, Vol. 4. Reading, MA: Addison-Weseley, 1989

18. M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991

19. Z. Manna, R. Waldinger: A Deductive Approach to Program Synthesis. ACM Trans. on Prog. Lang. and Sys. 2(1): 90-121 (1980)

20. J. Mostow: Design by Derivational Analogy: Issues in the Automated Replay of Design Plans. Artificial Intelligence 40, 119-184 (1989)

21 R. Paige, S. Koenig: Finite Differencing of Computable Expressions. ACM Transactions on Programming Languages and Systems 4(3): 402-454 (1982)

22. C.H. Papidimitriou, K. Steiglitz: Combinatorial Optimization. Englewood, N.J.: Prentice-Hall 1982

23. U.S. Reddy: Design Principles for an Interactive Program-Derivation System. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 453-482

24. C. Rich, R.C. Waters (eds.): Artificial Intelligence and Software Engineering. Los Altos: Morgan Kaufmann 1986

25. C. Rich: A formal representation for plans in the Programmer's Apprentice. ICJAI 1981

26. C. Rich, Y.A. Feldman: Seven Layers of Knowledge Representation and Reasoning in Support of Software Development. IEEE Trans. on Software Eng. 18(6), 451-469 (1992)

27. J.A. Robinson: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12(1), 23-41 (1965)

28. H.B. Rubenstein, R.C. Waters: The Requirements Apprentice: Automated assistance for requirements acquisition. IEEE Trans. on Software Eng. 17, 226-240 (1991)

29. D. Setliff: Synthesizing VLSI Routing Software from Specifications. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 207-226

30. D.R. Smith, M.R. Lowry: Algorithm Theories and Design Tactics. Science of Computer Programming 14:305-321 (1990)

31. D.R. Smith: KIDS -A Knowledge-Based Software Development System. In M.R. Lowry, R.D. McCartney (eds.): Automating Software Design. Cambride, MA: AAAI/MIT Press 1991, pp. 483-514

32. D.R. Smith: Constructing Specification Morphisms. Kestrel Institute Technical Report KES.U.92.1 (1992)

33. R.J. Waldinger, R.C. Lee: PROW: A step toward automatic program writing. IJCAI-69, pp. 241-252

34. R.C. Waters: Program Translation via Abstraction and Reimplementation. IEEE Trans. on Software Eng. 14(8), 1207-1228 (1988)

35. D.S Wile: Program Developments: Formal explanations of implementations. CACM 26(11): 902-911 (1983)

UPDATING LOGIC PROGRAMS

Nicola Leone, Luigi Palopoli[1] and Massimo Romeo
DEIS, Universita` della Calabria, 87036 Rende (CS), Italy.

Abstract. This paper proposes an update language for logic programming based knowledge systems. The language is built upon two basic update operators denoting insertions and deletions of positive literals (atoms), respectively. Several simple control structures have been defined by which basic updates can be combined to program complex updates. The presented approach is centered around the idea of executing a basic update operation by directly modifying the truth valuation of the (intensionally or extensionally defined) atom on which it is requested. Also the truth valuations of the atoms inductively depending on the updated one are accordingly modified. Several examples are presented which show that both deterministic and non-deterministic transformations of a logic program are easily expressed within the update language.

1 Introduction

Logic programming is today a well assessed tool for various kinds of advanced application domains of knowledge engineering including, for instance, intelligent database interfaces and expert systems. Well defined and efficient supports for query answering is surely the most important feature of a logic programming based knowledge system. However it is widely recognized that, in many applications, an adequate support to updating knowledge formalized in the logic program is needed [1, 18].

Roughly speaking, the problem of updating a logic program can be described as follows. Given a logic program P and an atom A to be inserted (resp., deleted), the task of updating is to modify the semantics of P in such a way that (1) the atom A is true (resp., false) in the modified semantics, and (2) the semantics is modified "consistently" with the rules in P. The latter point requires that a suitable form of "closure" is maintained within the modified semantics w.r.t. the inferences denoted by rules in P. The closure property we shall adopt is a very simple and intuitive one, and will be explained in the following.

In the recent years great attention has been paid to the problem of logic program updating (e.g., [7, 12, 13]). As a matter of fact, however, no solution to the problem of updating intensional knowledge can be found in the literature which can be regarded as being satisfactory in all applicative frameworks.

The first attempt to approach the problem of updating logic programs was made within the logic language Prolog [15]. Indeed, Prolog includes two built-in update operators, namely, *assert* and *retract*. However, these update operators have many semantic and operational faults, as lucidly pointed out by D.S. Warren in [17]. One major fault of these operators is the lack of a clear formal semantics.

To overcome these drawbacks several alternative proposals have been developed [3, 5, 6, 7, 10, 13, 14]. Most of them approach the problem of updating a logic program as a generalization of the view update problem in the relational database framework (e.g., [4]). These solutions, generally cleanly formalized, are based on "pushing-down" updates defined on intentional predicates towards base ones, because rules are regarded as certain, not updatable knowledge. However, even if there is strong evidence towards adopting the "push-down" philosophy in many

[1] Current affiliation is CS Department, UCLA, Los Angeles (CA), USA. Email: luigi@cs.ucla.edu.

situations, there are still a number of applications for which it is not well suited. A typical case in which a "push-down" semantics fails in providing suitable meaning to update activities arises when some general rule holds at a given time, but some of its specific instances is to be invalidated afterwards.

As an example, consider a knowledge base describing some aspects of the life in a zoo. A portion of the logic program constituting the knowledge base is shown next:

r_1: $eats(X,banana) \leftarrow monkey(X)$ $\quad$ $bear(yogi) \leftarrow$
r_2: $eats(X,honey) \leftarrow bear(X)$ $\quad$ $bear(bubu) \leftarrow$
r_3: $eats(X,salmon) \leftarrow bear(X)$ $\quad$ $monkey(kong) \leftarrow$
r_4: $happy(X) \leftarrow bear(X), eats(X,honey), eats(X,salmon)$
r_5: $slimming(X) \leftarrow bear(X), \neg eats(X,honey)$

The above logic program P_{zoo} describes the classification in species of the animals in the zoo, the foods each species is usually fed with and some further relationships between species, diet, happiness and weight modification of an individual animal. For instance, *yogi* is a bear and so he usually eats honey and salmon. Furthermore *yogi* is happy and he is not slimming. An exception to the ordinary diet occurs when, at some time, *yogi* the bear gets sick because he has eaten too much honey. In the new situation *yogi* is disallowed eating further honey. So we want a new semantics for the logic program in which the information that *yogi* can eat honey is no longer valid. In other words the atom $eats(yogi,honey)$ should not be part of the intended semantics of the logic program any longer, so that *yogi* is prevented eating further honey. Thus, from a *semantic viewpoint*, the expected result should be to make false the atom $eats(yogi,honey)$. Moreover, some form of "consistency" w.r.t. the logic program is to be provided, as atoms which are "no longer supported" by any ground instance rule are to be made false whereas atoms which are supported by some rule, and are not true in the current semantics, are to be made true. In the zoo example at hand, the "consistency" principle implies that the atom $slimming(yogi)$ is made true whereas the atom $happy(yogi)$ is made false, and the truth valuations of all other atoms remain unchanged.

Following a "pushing-down" approach the effect of the update is to delete $bear(yogi)$, which is clearly unintuitive as, even if sick, *yogi* is still a bear.

One could object that the expected result can be obtained, even if a "push-down" approach is assumed, by refining r_2 into:

$$eats(X,honey) \leftarrow bear(X), \neg ate_too_much_honey(X)$$

and then adding the fact $ate_too_much_honey(yogi)$ when needed. However, we note that, in a situation like the one we are describing here, we can have hundreds of causes determining hundreds of changing in the ordinary diet and to explicitly encode all the possibilities in the rules is obviously not convenient. It is also worth pointing out that the expected result can not be obtained by means of the Prolog *retract* operator, as it allows only to completely eliminate the rule r_2, henceforth enforcing all bears not to eat honey.

The *Update Logic Language* (*ULL*) presented in this paper is based on the idea that updates requested on an intensional predicate q have to be carried out by directly modifying the set of tuples on which q holds, rather than changing the extensions of base predicates on which q depends (in other words, no pushing down is performed). In order to maintain the consistency w.r.t. the implications defined in the program, along with the directly involved atoms, the truth values of atoms which are logical consequences of them have to be possibly modified as well (the atoms $happy(yogi)$ and $slimming(yogi)$ in the example at hand). In fact, the expected semantics of the example at hand is obtained by executing the ULL delete operation $-eats(yogi,honey)$.

ULL includes two basic update operators which are used to specify deletions (like the previous one) and insertion of atoms, respectively. Basic updates are not

expressive enough on their own, and so they are combined in ULL to form more complex updates as follows. First, several basic update operations can be sequenced by listing them one after another in the order in which they are intended to be executed. Second, it is possible to specify two conditions, each consisting of a logical goal (i.e., a possibly negated conjunction of literals), which influence the execution of an update. The former condition (precondition) is evaluated against the program semantics on which the update is requested; while the latter one (postcondition) is evaluated against the program semantics which would result from the application of the requested update operation. If both the conditions are satisfied then the update becomes persistent (i.e., the knowledge is modified according to the requested update operations). Otherwise the update is aborted and the logic program remains unchanged. Either conditions may be empty, in which case the trivial condition *True* is assumed. A further role played by pre- and post-conditions is to construct substitutions which are used to execute an update (see below). We notice that our notion of pre- post- conditions is a simplified form of the state-testing conditions of dynamic logics [8, 9]. Third, the programmer is given the possibility of choosing one out of three *execution modalities* under which an update is to be executed. In more details, the possible forms of an ULL update are the following:

a. $\exists (C_1\ U\ C_2)$, which denotes an update with an *existential* execution modality (*existential update*, for short);

b. $\forall (C_1\ U\ C_2)$, which denotes an update with a *universal* execution modality (*universal update*, for short);

c. $*(C_1\ U\ C_2)$, which denotes an update with an *iterated* execution modality (*iterated update*, for short).

where C_1 and C_2 are (possibly empty) goals, and U is an update operation of the form $u_1 u_2 \cdots u_n$, being u_i a basic update, $1 \leq i \leq n$ [2]

After its name, the execution modality determines the way an update is executed. Intuitively, an existential update $\exists (C_1\ U\ C_2)$ is executed by running $U\sigma$ on the logic program at hand, where σ is a substitution chosen non-deterministically amongst those making true the condition C_1 against the initial semantics and C_2 against the semantics obtained executing $U\sigma$. If no such substitution exists, then the existential update has a null effect. Consider universal update $\forall (C_1\ U\ C_2)$ and let Σ be the set of substitutions making true C_1 with respect to the initial semantics. The universal update has a non null effect only if for each $\sigma \in \Sigma$, $C_2\sigma$ is true with respect to the updated semantics obtained by executing $u_1\Sigma \ldots u_n\Sigma$, where $U = u_1 \ldots u_n$, in which case this updated semantics is the result of the update. (note that $u_i\Sigma$ denotes the execution of all the operations $u_i\sigma$ such that $\sigma \in \Sigma$). An iterated update $*(C_1\ U\ C_2)$ is executed by iteratively executing the existential update $\exists (C_1\ U\ C_2)$ until the semantics resulting from two successive application of this existential update yield an identical result. (or both) is no longer satisfiable against the respective logic program.

Finally, given k updates $\Psi_1 \ldots \Psi_k$ it is possible to specify the execution of them in sequence. To do this, ULL includes a sequential composition operator, denoted by ";".

The rest of the paper is organized as follows. Section 2 contains a number of examples of simple update problems and their solutions with *ULL* which should

---

[2] Note that the correct *ULL* syntax for the previous update *–eats(yogi,honey)* would have been either $\exists$ (*–eats(yogi,honey)*) or $\forall$ (*–eats(yogi,honey)*), or $\exists$ (*–eats(yogi,honey)*). However, as shall be apparent, the semantics of variable-free updates is independent on the execution modality. Therefore, in this case, w.l.o.g., we allow dropping the modality specifier.

help in informally illustrating the main characteristics of our approach. Then, Section 3 provides the formal definition of the update language *ULL*.

2 Sample Updates

In this section we shall present a number of examples which should shed some light on the structure of the *ULL* language. Some of the presented problems refer to updates requested on extensional predicates. However, we recall that our approach is based on the idea of making no difference between updating extensional predicates and updating intensional ones. Therefore, in our framework, the solutions to those problems, illustrated in the following, are completely "general". Before facing more involved update problems, let us give one more example of a very simple update.

Assume that, because of his disease, *yogi* the bear has temporarily to eat oranges. Then the logic program of the example can be update using $+eats(yogi,orange)$. When *yogi* eventually recovers his health, his diet can be reset to the ordinary one. This can be accomplished by

$$-eats(yogi,orange)\ +eats(yogi,honey).$$

Such an update, besides the two directly requested diet modifications (i.e., yogi does not eat oranges and he eats honey), also causes modifications of the semantics of the predicates depending on *eats*. In particular, yogi is again happy and not slimming. Hence, the original semantics of P_{zoo} is completely restored. We note that the previous update consists of a sequence of two operations, which are executed one after another from left to right in the order in which they appear.

As already stated, the *ULL* language allows to use different execution modalities for updates. Let us start with the $\forall$ modality. For an example of it, assume that the zoo has several employees some of which look after the animals. Each employee is characterized by his name, his seniority, expressed in years, and his monthly salary, expressed in dollars. Furthermore, employees looking after bears receive an extra salary of 100 dollars per month. So, assume the following facts belong to the logic program.

$employee(tom,1,1500)\leftarrow$ $\qquad$ $look_after(tom,kong)\leftarrow$
$employee(john,2,2000)\leftarrow$ $\qquad$ $look_after(john,yogi)\leftarrow$
$employee(bob,3,2500)\leftarrow$ $\qquad$ $look_after(bob,bubu)\leftarrow$
$extra_pay(X,100)\leftarrow bear(Y), look_after(X,Y)$

To refer to a classical database example, assume that the salary of all the employees having been working at the zoo for at least two years is to be raised by five percent. Then, a set oriented update of the logic program can be executed, which is expressed as follows in *ULL*:

$$\forall(employee(X,Z,Y),\ Z\geq 2\ -employee(X,Z,Y)\ +employee(X,Z,Y\times 1.05))$$

Intuitively, the above update works as follows: first, the set Σ of the substitutions satisfying the conjunction $employee(X,Z,Y),\ Z\geq 2$ is computed (i.e., the set of all employees with a working seniority greater than two years is determined); second, for each substitution in Σ, the update operation $-employee(X,Z,Y)\sigma$ is executed (i.e., the old employee's tuple is removed); finally, for each σ in Σ, the update operation $+employee(X,Z,Y\times 1.05)\sigma$ is executed (i.e., the new employee's tuple, including his raised salary, is inserted). Since no postcondition is present, the update succeeds and the modification becomes persistent. In order to show the use of postconditions, assume that the increase of the employees' salaries has to be executed only if it does not cause that an employee with two years of seniority earns more than 2100 dollars for his base salary. Such a constraint on the execution of the update can be imposed by simply adding the postcondition $\neg(employee(X,2,T),\ T>2100)$ to the previous update:

$$\forall (\ employee(X, Z, Y), Z \geq 2\ -employee(X, Z, Y)$$
$$+employee(X, Z, Y \times 1.05)\ \neg(employee(X, 2, T), T > 2100))$$

Thus, in the case of a universal updates, the precondition determines the set of substitution on which the update operation is to be executed (the employees whose salary has to be raised, in the example at hand), whereas the *postcondition* determines whether the update has to be actually executed or not (it is a sort of constraint on the whole execution of the update). By the way, notice how naturally and compactly *ULL* allows to express the above updates.

As already mentioned, *ULL* allows to specify iteration of tuple-oriented updates. Iteration is necessary in order to endow the update language with a good expressive power - we notice that we do not have recursion as, for instance the top-down language in [12], or the bottom-up language in [2]. For an example of an iterated update, assume that we want to level-up the salary of employees with the same seniority. This is accomplished by the following update:

$$*\ (employee(X, Y, Z), employee(X', Y, Z'), X \neq X', Z < Z',$$
$$-employee(X, Y, Z), +employee(X, Y, Z'))$$

where the single leveling-up step $-employee(X, Y, Z), +employee(X, Y, Z')$ is iterated while there exists a pair of employees with the same seniority Y but different salaries. It should be clear that the execution of an iterated update can be non-terminating. However, this is the same situation one has to handle using *while* loops in conventional programming languages. In any case, some time-out mechanism could be provided if necessary.

For other instances of ULL updates let us switch away from the zoo example and consider the following logic program P_{graph} defining the direct graph G_1 (shown in Figure 2.1) along with its transitive closure.

$$arc(c, a) \leftarrow \qquad arc(a, b) \leftarrow$$
$$r_1: \quad tc(X, Y) \leftarrow arc(X, Y) \qquad arc(b, a) \leftarrow$$
$$r_2: \quad tc(X, Y) \leftarrow arc(X, Z), tc(Z, Y) \qquad arc(b, c) \leftarrow$$

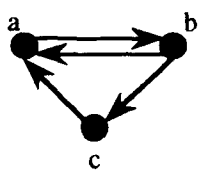


Fig. 2.1

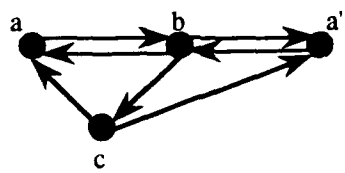


Fig. 2.2

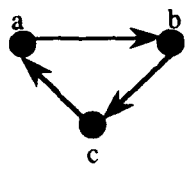


Fig. 2.3

Assume we want to create a "duplicate" of the node a in the graph, i.e., a node a' which has the same sets of incoming and outgoing arcs as a. This is obtained by:

$$\forall (arc(a, X) + arc(a', X));\ \forall (arc(X, a) + arc(X, a'))$$

Note that the above update consists of two updates, $\forall (arc(a, X) + arc(a', X))$ and $\forall (arc(X, a) + arc(X, a'))$, combined through a sequence control structure, denoted in ULL by ";". The resulting graph is shown in Figure 2.2. It is worth pointing out once again that, according to ULL semantics (see Section 3), the truth valuation associated with the intensional predicate tc is consequently modified. For instance, the atom $tc(a', c)$ becomes true after update execution.

An "orientation" of a graph G is a maximal subgraph of G containing no pairs of symmetric arcs. The following ULL update orients the graph G_1.

$$*\ (arc(X, Y), arc(Y, X)\ -arc(X, Y))$$

The previous update is non-deterministic. One of its results is shown in Figure 2.3.

We remark the importance of non-determinism encoded in the language. Indeed

an orientation of the graph cannot be obtained by a deterministic language not allowing "pick-up-one" choices of indistinguishable data [2].

3 The Update Logic Language

The section makes use of some basic notions concerning logic programming and well-founded semantics. For room's reasons such notions are not reported here. The reader may refer to [11, 16] for a detailed presentation of these topics.

Given a logic program P, we assume the *well-founded* model of P, denoted $WF(P)$, as the intended meaning of a logic program P. Thus, given a ground positive goal G (resp., negative goal $\neg(G)$), we say that P *implies* G (resp., P *implies* $\neg(G)$), denoted $P \vdash G$ (resp., $P \vdash \neg(G)$), if and only if G is true (resp., G is false) with respect to $WF(P)$.

The sequel of this section formally presents both the syntax and the semantics of *ULL*.

3.1 Syntax

The primitive manipulations definable on a logic program are insertions and deletions of atoms. Insertions and deletions are denoted by *update operators* + and −, respectively. Thus, a *simple update operation* is a syntactic of the form $\otimes Q$, where $\otimes \in \{+, -\}$ is an update operator, and Q is a an atom. If Q is ground then $\otimes Q$ is a *ground simple update operation.*

In particular, $+Q$ is called *insertion*, while $-Q$ is called *deletion*.

Simple updates are combined in sequences to express update operations involving more atoms. Let $\{u_1,...,u_n\}$ denote a group of simple updates. Then $u_1...u_n$ is an *update operation.* If $\{u_1,...,u_n\}$ are all ground then $u_1...u_n$ is a *ground update operation.* Intuitively, the execution of the update operation $u_1...u_n$ corresponds to execute the simple update operation u_1 first; then, u_2 is fired and so on, until to the execution of the simple update u_n.

The above defined update operations are the basic primitives for modifying the meaning of a program. However, many real world situations require that an update is executed only if some conditions are verified. The general updates that we define next allows us to deal with such situations by specifying a precondition and a postcondition to the update execution. In order to endow our language with a good flexibility and expressibility, we introduce three different types of updates along with a composition operator which allows to execute updates in sequence.

Definition. Let C_1, C_2 denote two goals, and U be an update operation. Then

1. $\exists(C_1\ U\ C_2)$ is a *general update* (called *existential* update);
2. $\forall(C_1\ U\ C_2)$ is a *general update* (called *universal* update);
3. $*(C_1\ U\ C_2)$ is a *general update* (called *iterated* update);
4. $\Psi_1;\Psi_2$ is a *general* update (called *compound* update) if Ψ_1 and Ψ_2 are general updates.

3.2 Formal Semantics

The core idea of our approach can be rephrased as follows:

i. the insertion $+Q$ of a ground atom Q modifies the semantics of P by adding to its well-founded model $WF(P)$ the atom Q along with its logical consequences (according to P);

ii. the deletion $-Q$ of a ground atom Q discards from $WF(P)$ the atom Q and all those atoms which have been inductively inferred from Q.

Since the semantics of a logic program is completely determined by its instantiation, we regard an update on P as a transformation of $ground(P)$. Indeed it is easily seen that for each program P, $WF(P)=WF(ground(P))$. Then, the meaning of the program after the execution of the update will be the well-founded model of its transformed instantiation. Thus, the semantics of updates will be given by specifying a mapping ϕ which takes a ground program $ground(P)$ and a general update Ψ as the operands and returns the (ground) program resulting from updating $ground(P)$ by Ψ. The transformation ϕ is defined in terms of another function τ which takes a ground program $ground(P)$, an update operation U and a family Σ of substitutions as its operands and returns a resulting (ground) program $\tau(ground(P), U, \Sigma)$. From now on, and throughout this section, we assume that a logic program P has been fixed and we denote its instantiation $ground(P)$ by $\bar{P}$. The mapping τ is inductively defined as follows.

if $U = +Q$, then $\tau(\bar{P}, +Q, \Sigma) = \bar{P} \cup \{ \sigma(Q) \leftarrow \mid \sigma \in \Sigma \}$;

if $U = -Q$, then $\tau(\bar{P}, -Q, \Sigma) = \bar{P} - \{ r \in \bar{P} \mid \exists \sigma \in \Sigma \text{ s.t. } H(r) = \sigma(Q) \}$;

if $U = \otimes_1 Q_1 \cdots \otimes_k Q_k$ $(k \geq 2)$, then
$\tau(\bar{P}, \otimes_1 Q_1 \cdots \otimes_k Q_k, \Sigma) = \tau(\tau(\bar{P}, \otimes_1 Q_1, \Sigma), \otimes_2 Q_2 \cdots \otimes_k Q_k, \Sigma))$.

Now, we can define the semantics of updates. The first update we consider is the existential one.

Definition. An existential update $\exists (C_1 \ U \ C_2)$ is *applicable* if there exists a substitution σ such that: (i) $\bar{P} \vdash \sigma(C_1)$, and (ii) $\tau(\bar{P}, U, \{\sigma\}) \vdash \sigma(C_2)$.
If $\exists (C_1 \ U \ C_2)$ is applicable then $\phi(\bar{P}, \exists (C_1 \ U \ C_2)) = \tau(\bar{P}, U, \{\sigma\})$, where σ is any substitution verifying the conditions (i) and (ii) above; otherwise, $\phi(\bar{P}, \exists (C_1 \ U \ C_2)) = \bar{P}$. □

Thus, the execution of an existential update, say, $\exists (C_1 \ U \ C_2)$, corresponds to the execution of the ground update operation σU, where σ is a substitution such that the precondition σC_1 is verified in $\bar{P}$, and the postcondition σC_2 is true in the program transformed by the update operation σU. It is worth noting that σ is non-deterministically chosen in the set of substitutions satisfying such conditions. We notice that in the case either of the conditions is empty, the trivial condition *True* is assumed. Therefore, if C_1 is empty, the update is executed along with *any* substitution satisfying C_2 in the resulting program. If C_2 is empty, then the update is executed along with any substitution satisfying C_1 in the logic program.

The examples of this section will refer to the logic program P_{graph}, defining the direct graph G_1, introduced in Section 2.

Example To eliminate one of the arcs ending at a node Y such that the arc (b, Y) is in the graph, we execute the general update:

$$\exists (arc(b, Y), arc(X, Y), -arc(X, Y)).$$

Here, the substitutions verifying the requested precondition are $\sigma_1=\{Y/a, X/c\}$, $\sigma_2=\{Y/c, X/b\}$, and $\sigma_3=\{Y/a, X/b\}$[3]. Since no postcondition has to be verified, then all such substitutions are eligible for instantiating the update operation. Hence, the possible values of

$$\phi(ground(P_{graph}), (\exists (arc(b, Y), arc(X, Y) - arc(X, Y)))),$$

which represents the result of the execution of the above update on $ground(P_{graph})$ are:

$\bar{P}_1 = \{arc(a, b), arc(b, a), arc(b, c)\} \cup TC$,
$\bar{P}_2 = \{arc(a, b), arc(b, a), arc(c, a)\} \cup TC$, and,

[3] Note that, even if a substitution is a mapping from V to B_P, we have specified only the assignments of the variables appearing in the update, as the values assigned to the other variables does not affect the semantics of the update.

$$\bar{P}_3 = \{arc(a, b), arc(b, c), arc(c, a)\} \cup TC$$

where TC is the set of all ground instances of the two rules defining the predicate symbol tc (which remains unchanged).
If one wishes to avoid deleting the arcs starting from the node b he has simply to add the condition $X \neq b$ in the precondition. In this case the execution is deterministic, as the only applicable substitution is σ_1; so, the result of the execution yields $\bar{P}_1$. On the other hand, if one wants that, after the execution of the update, some arc must still reach the node c, then the general update can be modified by adding the postcondition $arc(Z, c)$:

$$\exists(arc(b, Y), arc(X, Y) - arc(X, Y)\ arc(Z, c).$$

In this case the substitution σ_2 is not applicable; thus the result of the execution is either $\bar{P}_1$ or $\bar{P}_2$. □

Now, let us turn our attention to universal updates. Given an universal update

$$\forall(C_1\ U\ C_2)$$

we denote by Σ_{C_1} the family of all substitutions satisfying C_1, i.e., $\Sigma_{C_1} = \{\sigma \mid \sigma \text{ is a substitution and } \bar{P} \vdash \sigma(C_1)\}$.

Definition. An universal update $\forall(C_1\ U\ C_2)$ is *applicable* if, for each $\sigma \in \Sigma_{C_1}$, $\tau(\bar{P}, U, \Sigma_{C_1}) \vdash \sigma(C_2)$.
If $\forall(C_1\ U\ C_2)$ is applicable then $\phi(\bar{P}, \forall(C_1\ U\ C_2)) = \tau(\bar{P}, U, \Sigma_{C_1})$;
otherwise, $\phi(\bar{P}, \forall(C_1\ U\ C_2)) = \bar{P}$. □

Hence, in case of an universal update, say, $\forall(C_1\ U\ C_2)$, the update operation σU is executed for *each* substitution σ such that σC_1 is verified in $\bar{P}$. However, if, for some substitution $\sigma \in \Sigma_{C_2}$, σC_2 is not true in the obtained program, then the whole execution of the update is aborted (i.e., the update produces a null effect).

Example Let consider the 'universal version' of the update shown in the previous example:

$$\forall(arc(b, Y), arc(X, Y) - arc(X, Y)),$$

which means: delete *all* arcs ending at a node Y such that the arc (b, Y) is in the graph. The substitutions verifying the applicability condition are the same as in the previous case: $\sigma_1 = \{Y/a, X/c\}$, $\sigma_2 = \{Y/c, X/b\}$, and $\sigma_3 = \{Y/a, X/b\}$. Hence, $\Sigma_{arc(b, Y), arc(X, Y)} = \{\sigma_1, \sigma_2, \sigma_3\}$. Since a universal execution modality has been specified, the update operation $-arc(X, Y)\sigma$ is executed on each $\sigma \in \Sigma_{arc(b, Y), arc(X, Y)}$. Thus, the result $\phi(ground(P_{graph}), (\forall (arc(b, Y), arc(X, Y) - arc(X, Y))))$ of the update is $\{arc(a, b)\} \cup TC$ which agrees with the given intuition. As we already pointed out in Section 2, in a universal update the postcondition plays the role of a constraint which, if not satisfied, invalidates the whole update execution. For instance, let us add to the previous update the postcondition $tc(a, a)$ by which we are requesting that, after the execution of the update, the node a is in a cycle:

$$\forall(arc(b, Y), arc(X, Y) - arc(X, Y)\ tc(a, a).$$

Now, the set $\Sigma_{arc(b, Y), arc(X, Y)}$ is again $\{\sigma_1, \sigma_2, \sigma_3\}$. Thus, $\tau(ground(P_{graph}), \{\sigma_1, \sigma_2, \sigma_3\}, -arc(X, Y))$ is $\bar{P}' = \{arc(a, b)\} \cup TC$ as before. However, the postcondition $tc(a, a)$ is not satisfied in $\bar{P}'$. Hence, the update is aborted and $\phi(ground(P_{graph}), (\forall(arc(b, Y), arc(X, Y) - arc(X, Y) tc(a, a))))$ $= ground(P_{graph})$, i.e., the program remains unchanged. Finally it is worth noting that, in a universal update, the range of the variables is determined by the precondition; so, a variable appearing only in the postcondition is universally quantified. For instance if we change $tc(a, a)$ into $tc(a, Z)$ in the previous update, then the postcondition express the requirement that $tc(a, Z)$ is true for each Z, with Z ranging in the Herbrand Universe. On the contrary, since the variable X appears also

in the precondition, a postcondition $tc(a,X)$ would require only the truth of $tc(a,b)$ and $tc(a,c)$, as b and c are the values for X in σ_1, σ_2 and σ_3. □

Finally, let us consider the iterated update.

Definition. An iterated update $*(C_1\ U\ C_2)$ is *applicable* if there exists a substitution σ such that: (i) $P \vdash \sigma(C_1)$, (ii) $\tau(P, U, \{\sigma\}) \vdash \sigma(C_2)$, and, (iii) $\tau(P, U, \{\sigma\}) \neq P$. If $*(C_1\ U\ C_2)$ is applicable, then $\phi(P, *(C_1\ U\ C_2)) = \phi(\tau(P, U, \{\sigma\}), *(C_1\ U\ C_2))$, where σ is any substitution verifying the conditions (i), (ii), and (iii) above; otherwise, $\phi(P, *(C_1\ U\ C_2)) = P$. □

Thus, executing an iterated update $*(C_1\ U\ C_2)$ corresponds to repeatedly applying the existential update $\exists(C_1\ U\ C_2)$ until it becomes not applicable. The further condition (*iii*) on applicability discards those substitutions which, though making the pre- and post- conditions true, do not actually cause the program to be modified.

Example. Let consider the iterated update, discussed in Section 2, which orients the graph of program P_{graph}:

$$*(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y))$$

It is easy to recognize that the only applicable substitutions are $\sigma_1=\{X/a, Y/b\}$ and $\sigma_2=\{Y/a, X/b\}$. Hence, the effect of the first iteration is to delete either the arc (a,b) or the arc (b,a) from the graph. Then, in either case no substitution is applicable at the second round (i.e., on the modified program), so the execution terminates. Formally, let $P_1=ground(P_{graph})-\{arc(a,b)\}$ and $P_2=ground(P_{graph})-\{arc(b,a)\}$, then there are two possible values for $\phi(ground(P_{graph}), *(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y)))$:

either the chosen substitution is σ_1, then

$$\phi(ground(P_{graph}), *(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y))) = \phi(P_1, *(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y))) = P_1;$$

or the chosen substitution is σ_2, then

$$\phi(ground(P_{graph}), *(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y))) = \phi(P_2, *(arc(X,Y), arc(Y,X) \text{ -} arc(X,Y))) = P_2 \ \square$$

As already mentioned, in general, we are not guaranteed that the execution of an iterated update terminates in any case.

Example. The update $*(arc(X,Y), \text{-}arc(X, f(Y)) + arc(X, f(Y))$ goes on adding new nodes and arcs to the example graph never terminating. □

For ground updates, we have the following:

Proposition. *Let P be a logic program, U be a ground update operation, and, G_1 and G_2 be two ground goals. Then*

$$\phi(ground(P), \exists(G_1 U G_2)) = \phi(ground(P), \forall(G_1 U G_2)) = \phi(ground(P), *(G_1 U G_2)) \ \square$$

The above result justifies dropping the modality specifier for variable free general updates, as we have often done in the previous sections.

We have described how the execution of one single general update affects a ground program. It is easily recognized that the approach immediately extends to sequences of general updates. Indeed, the ϕ function yields programs, and therefore it can be reapplied on its results. As a consequence, the semantics of a sequence of n general updates is given by applying the ϕ function n times. More formally, a sequence of general updates to be applied on a program P is seen as a unique compound update whose semantics is defined next.

Definition. Given a logic program P, let $\Psi_1; \cdots ; \Psi_n$ be a compound general update. Then

$$\phi(ground(P), \Psi_1; \cdots ; \Psi_n) = \phi(\phi(ground(P)), \Psi_1)), \Psi_2; \cdots ; \Psi_n)) \ \square$$

Obviously, if one of the general updates composing the sequence is an iterated

update and it goes in a never ending loop, also the entire sequence loops indefinitely.

We are finally in the position of giving the definition of the semantics of an updated program against which to perform query answering.

Definition. Let P be a logic program and let $\Psi = \Psi_1, \ldots, \Psi_n$ be a list of general updates to be executed on P. The semantics $S_{P,\Psi}$ of the program obtained updating P by Ψ is defined as the well founded model of $\phi(ground(P), \Psi))$ if $\phi(ground(P), \Psi)$ is defined, and it is undefined, otherwise.

References

1. Abiteboul, S., Updates, a new frontier, in: Proc. Second Int. Conf. on Database Theory, LNCS 326, Springer-Verlag, 1-18, 1988.
2. Abiteboul, S. and V. Vianu, Datalog extensions for database queries and updates, JCSS, 43(1), 62-124, 1991.
3. Atzeni, P. and R. Torlone, Updating intensional predicates in datalog, Data and Knowledge Engineering, 1992.
4. Bancilhon, F., and N.Spyratos, "Update semantics of relational views", *ACM TODS*, 6(40), Dec. 1981.
5. Decker, H., Drawing updates from derivations, in: Proc. 3th Int. Conf. on Database Theory, LNCS 460, Paris, 1990.
6. Grahne, G., A.O. Mendelzon and P.Z. Revesz, Knowledgebase transformations, in: Proc. 11th ACM PODS, 246-260, 1992.
7. Guessoum A. and J.W. Lloyd, Updating knowledge bases, New Generation Computing, 8(1), 71-89, 1990.
8. Harel, D., "First-order dynamic logic", LNCS (Goos, G., and J.Hartmains eds.), Springer-Verlag, 1979.
9. Harel, D., "Dynamic logic", in Handbook of Philosophical Logic, (Gabbay and Guenther, eds.), D.Reidel Publishers, 1983.
10. Kakas, T. and P. Mancarella, Database updates through abduction, in: Proc. 16th Int. Conf. on Very Large Databases (Morgan-Kaufmann, Los Altos, CA, 1990) 650-661.
11. Lloyd, J.W., Foundations of logic programming (Springer, Berlin, 2nd ed., 1987).
12. Manchanda, S. and D.S. Warren, "Towards a logical theory of database view updates", Int.Worksh. on Foundations of Deductive databases and Logic Programming, J.Minker ed., Aug. 1988.
13. Naqvi,S., and R.Krishnamurthy, "Database updates in logic programming, ACM PODS, 1988.
14. Rossi,F., and S.Naqvi, "Contribution to the view update problem", Proc. Int. Conf. on Logic Programming, 388-415, 1989.
15. Sterling, L. and E. Shapiro, The art of Prolog, MIT Press, Cambridge, 1986.
16. Van Gelder, Ross, Schlipf, The well-founded semantics of general logic programs, *Journal of ACM*, 38(3), 620-650, 1991.
17. Warren, D.S., Database update in pure Prolog, in: Proc. Int. Conf. on Fifth Generation Computer Systems, 244-253, 1985.
18. M. Winslett, "A model-theoretic approach to updating logical databases", ACM PODS, 1986.

Expressing Program Requirements using Refinement Lattices

Dave Robertson †, Jaume Agustí ‡, Jane Hesketh †, Jordi Levy ‡
†Department of Artificial Intelligence, University of Edinburgh.
‡IIIA, Centre d'Estudis Avançats de Blanes, Blanes, Spain.

Abstract

Requirements capture is a term used in software engineering, referring to the process of obtaining a problem description – a high level account of the problem which a user wants to solve. This description is then used to control the generation of a program appropriate to the solution of this problem. Reliable requirements capture is seen as a key component of future automated program construction systems, since even small amounts of information about the type of problem being tackled can often vastly reduce the space of appropriate application programs. Many special purpose requirements capture systems exist but few of these are logic based and all of them operate in tightly constrained domains. This paper introduces a formal language for requirements capture which bridges the gap between an order sorted logic of problem description and the Prolog programming language. An extended version of this paper appears in [4].

1 Introduction

Previous work on requirements capture, described in [5], attempted to control the generation of Prolog programs by applying domain knowledge from a problem description supplied by the user. This approach is attractive because it buffers users from part of the programming task. However, there is a tension between the demands of users for a notation to which they can relate and the need for computational sophistication in their application programs. This tends to create a conceptual gap between the languages of problem description and application. The trade–offs which were made in attempting to bridge this gap are discussed in [6] but the end–result is normally that the language used for problem description is different from the language used to describe the application program. This can become a serious problem if the means by which the two languages interact during program generation is not well understood.

One way to tackle these problems is to devise a language which can be used for problem description but also has a straightforward translation to an application programming language. This language has to be expressive but it must also be easy to use. In addition, it should be capable of describing a programming problem in general terms or in greater detail, depending on users' preferences. Previous work by Bundy and Uschold ([1]) has attempted to provide this sort of uniform language based on typed lambda calculus but they have yet to implement these ideas in a working system and the complexity of the mathematics involved makes it difficult to see how users without specialist training could feel confident about using it. A solution to this problem would be to "dress up"

the mathematics in a form which is more easily understood. Unfortunately, it is often difficult to make an inherently complex notation appear *simple*. An alternative, which we adopt in this paper, is to start with comparatively simple underlying principles and to manipulate these to obtain complex programs. A good source of ideas for this approach is in logic programming, which (in the form of pure Prolog programs) embodies a simple but powerful programming paradigm. A second source of inspiration is to be found in recent set-based specification languages. In particular we have drawn upon ideas from the COR system of refinements ([2]).

The core of the requirements capture language depends on representing a lattice of sets of results of predicates. This constitutes our problem description language. Section 2 introduces this notion in the context of Prolog[1]. This is followed, in Section 3 by a description of the way in which expressions in the language may be translated into Prolog. Since this is intended to be a high level language, not all of the axioms translate directly into Prolog and some are used, with the aid of proof rules, to control problem description. In Section 4 we describe some of the proof rules which we use later, in Section 5, to provide guidance in defining set lattices. Finally, in Section 6, we describe how programs (at differing levels of detail) may be extracted from our lattices.

2 Denoting Argument Sets

It is conventional to define the meaning of a logic program to be the set of ground unit goals deducible from that program. This gives a form of "global" meaning for a predicate in terms of all its arguments but it is possible to define more local interpretations in terms of stipulated arguments. We shall use the notation $V : P$ to denote the set of instances for the variable V which can be obtained from the goal P. In order sorted logics, it is normal to restrict the range of objects over which variables in formulae are permitted to range. We can achieve this effect using our notation by permitting the variables inside the goal expression to be restricted using the ':' operator. This permits any predicate to be applied over sets of objects, rather than over individuals as would be the case in standard first–order predicate calculus. The interpretation of a predicate argument applied in this way is defined as the set of results for the variables on the left of the ':' operator, given the application of the predicate to every combination of elements in the sets denoted in its arguments. Since all of our terms represent sets of objects we can introduce some standard set operators as follows:

Definition 1 *If A and B are set expressions then $A \cap B$ is the intersection of A and B; $A \cup B$ is the union of A and B; and $A \supseteq B$ denotes that B is a subset of A.*

[1]Throughout this paper we shall be using "pure" Prolog, without complicating features such as cut or side–effecting predicates

The use of the $\supseteq$ operator allows us to arrange our set expressions into a lattice. To provide a "top" and "bottom" to this lattice we shall use the symbol $\top$ to denote the entire universe of discourse and $\bot$ to denote the empty set of objects. The full syntax of refinement expressions appears below:

Definition 2 *A refinement formula is of the form $H \supseteq B$, where:*

- *H is the head of the refinement and is a primitive set expression.*
- *B is the body of the refinement and can be any set expression.*
- *A primitive set expression is of the form $V : P$, where V is a variable appearing in P and P is one of the following:*
  - *A Prolog goal.*
  - *A term of the form $Q(A_1, \cdots, A_N)$, where Q is a predicate name and each A_I is either a variable, constant or set expression.*
- *A set expression is of the form $V : E$, where V is a variable appearing in E and E is one of the following:*
  - *A primitive set expression.*
  - *A union of set expressions, $V_1 : E_1 \cup V_2 : E_2$*
  - *An intersection of set expressions, $V_1 : E_1 \cap V_2 : E_2$*
  - *The difference between two set expressions, $V_1 : E_1 - V_2 : E_2$*

  V is said to be restricted by the expression E. Any variable which is not restricted in this way is said to be unrestricted.
- *Any unrestricted variable in the head of a refinement formula must appear in the body of that formula.*

The next section will make more clear why the restrictions on syntax supplied in definition 2 are needed. It is worth noting in passing that set expressions for first order predicate calculus have also been introduced in [3] but in a different form and for different purposes.

3 Mapping Prolog to the Refinement Language

Section 2 introduced the basic notation for the refinement language. The purpose of this section is to show how the language can be understood in terms of Prolog. To simplify our explanation, we shall demonstrate the correspondence for unary predicates but the same principles apply to predicates of any arity. The $\supseteq$, $\cap$, and $\cup$ operators can be interpreted straightforwardly in terms of the logical connectives for implication ($\leftarrow$), conjunction (&) and disjunction ($\vee$). The correspondences are as shown below:

Set expression	FOPC formula
$V_1 : P(V_1) \supseteq V_2 : Q(V_2)$	$P(V) \leftarrow Q(V)$
$V_1 : P(V_1) \cap V_2 : Q(V_2)$	$P(V)$ & $Q(V)$
$V_1 : P(V_1) \cup V_2 : Q(V_2)$	$P(V) \vee Q(V)$

Any nested variable restrictions (using the ':' operator) within terms must be converted into preconditions for logical rules. Thus, if we have an expression of the form:

$$V_1 : P(V_1) \quad \supseteq \quad V_2 : Q(V_3 : A(V_3), V_2) \tag{1}$$

we would rewrite it to the expression:

$$P(V) \quad \leftarrow \quad A(V_3) \,\&\, Q(V_3, V) \tag{2}$$

It is important to remember that not all the refinement formulae are intended to translate directly into Prolog. In general, the refinement relation is more "permissive" than standard implication and with it we can represent a wide variety of information, only part of which is sufficiently precise to constitute a Prolog program. In particular, it is not always possible to translate from refinements which have restricted variables in the head but these variables do not appear in the body. Since our refinement language is, in this sense, very flexible we must be careful which axioms are allowed to be translated into Prolog. However, provided such checks are in place, we can benefit from the extra flexibility during problem description. For this, we need to use some standard proof rules, which are the topic of the next section.

4 Refinement Proof Rules

Since all the expressions in the language refer to sets, we can use proof rules from set theory to perform many of the operations necessary during requirements capture. This section describes some of the proof rules which we currently use and we anticipate that further, derived rules will be added to the collection as the system matures – for instance, rules describing the preservation of unions and intersections of predicates and a full set of rules for the set difference operator. In subsequent sections we shall show some of these rules in operation. In the proof rules which follow, the symbols A, B and C denote set expressions; $\top$ denotes the universal set; and $\bot$ denotes the empty set.

Proof rule 1 $\vdash \top \supseteq A$

Proof rule 2 $\vdash A \supseteq \bot$

Proof rule 3 $\vdash A \supseteq A$

Proof rule 4 $A \supseteq B, B \supseteq C \vdash A \supseteq C$

Proof rule 5 $C \supseteq A, C \supseteq B \vdash C \supseteq A \cup B$

Proof rule 6 $\vdash A \cup B \supseteq A$ *and* $\vdash A \cup B \supseteq B$

Proof rule 7 $\vdash A \supseteq A \cap B$ *and* $\vdash B \supseteq A \cap B$

Proof rule 8 $B \supseteq A, C \supseteq A \vdash B \cap C \supseteq A$

Proof rule 9 $\vdash (A \cap B) \cup (A \cap C) \supseteq A \cap (B \cup C)$

Proof rule 10 $A \supseteq A' \vdash V : P(A) \supseteq V' : P(A')$

5 Defining a Refinement Lattice

It would be possible to define complete programs entirely within the refinement language. However, this doesn't seem to us to be the most advantageous use of the language, since it merely replicates a standard logic program. In defining refinement lattices a key idea is that people should be allowed to "rise above" the level of the application program in the initial stages of refinement. When performing this task, it is common to want to combine existing ref. ments in order to be more specific about the way in which they apply. To support this process we permit users to restrict the size of a refinement expression on either (or both) the left or right sides. Since this could result in an overdefined expression – for example by over–restricting the left–hand side of the refinement – we must also apply a test for overdefinition to the resulting expression (described in [4]).

Definition 3 *A refinement of the form* $A \supseteq B$ *is an extension of the refinement lattice* $\mathcal{H}$ *if* $A' \supseteq B' \in \mathcal{H}$ *and* $A' \supseteq A$ *and* $B' \supseteq B$.

For example, we might have added the information that locations of fish include aquatic habitats; that aquatic habitats include rivers and that Carp are fish:

$$H : aquatic_habitat(H) \supseteq L : location(F : fish(F), L) \quad (3)$$

$$H : aquatic_habitat(H) \supseteq R : river(R) \quad (4)$$

$$F : fish(F) \supseteq C : carp(C) \quad (5)$$

Now if we add the information that the locations of Carp include rivers:

$$R : river(R) \supseteq L : location(C : carp(C), L) \quad (6)$$

we can show that this is a valid extension as follows:

- By definition 3 using axiom 3, we have an extension if:
  $L : location(F : fish(F), L) \supseteq L : location(C : carp(C), L)$ and
  $H : aquatic_habitat(H) \supseteq R : river(R)$
- By proof rule 8 we can establish that:
  $L : location(F : fish(F), L) \supseteq L : location(C : carp(C), L)$ if
  $F : fish(F) \supseteq C : carp(C)$.

- $F : fish(F) \supseteq C : carp(C)$ by proof rule 4 using axiom 5.
- $H : aquatic_habitat(H) \supseteq R : river(R)$ by proof rule 4 using axiom 1.

We would like, as far as possible, to protect users against including refinements which are overdefined within the existing lattice. We use the symbol, $\perp$, as the empty set expression and assume that all new sets added will be (potentially) larger than $\perp$. Therefore:

Definition 4 *A refinement, $A \supseteq B$ is overdefined if, in conjunction with the other axioms of the existing refinement lattice, $A \supseteq B \vdash \perp \supseteq B$.*

One of the main purposes of this definition is in limiting the ways in which set expressions can be refined, thus reducing the range of choices available to users when constructing the lattice. For example, it is sometimes useful to be able to define a predicate which can range over only particular sets of arguments but not others. For example, we might want to say that spiders only eat living things. We can express this using the axiom:

$$X : living(X) \supseteq X1 : eats(S : spider(S), X1) \tag{7}$$

If we also add the constraint that nothing is both living and dead:

$$\perp \supseteq X1 : living(X1) \cap X2 : dead(X2) \tag{8}$$

then we can protect against generalisation of the $eats/2$ predicate. For example, if we try to add the axiom:

$$X : eats(S : spider(S), X) \supseteq X1 : dead(X1) \tag{9}$$

we can prove that this is overdefined as follows:

- By proof rule 4, $\perp \supseteq X : dead(X)$ if:
  $\perp \supseteq X1 : living(X1) \cap X2 : dead(X2)$ (Axiom 8) and
  $X1 : living(X1) \cap X2 : dead(X2) \supseteq X : dead(X)$
- By proof rule 8, $X1 : living(X1) \cap X2 : dead(X2) \supseteq X : dead(X)$ if:
  $X1 : living(X1) \supseteq X : dead(X)$ and
  $X2 : dead(X2) \supseteq X : dead(X)$
- By proof rule 4, $X1 : living(X1) \supseteq X : dead(X)$ if:
  $X1 : living(X1) \supseteq X2 : eats(S : spider(S), X2)$ (Axiom 7) and
  $X2 : eats(S : spider(S), X2) \supseteq X : dead(X)$ (Axiom 9)
- By proof rule 3, $X2 : dead(X2) \supseteq X : dead(X)$

5.1 A Simple Example

Having defined mechanisms for creating and extending refinements we introduce, in this section, a short example to demonstrate the way in which the language may be used to develop incrementally a requirements specification. We shall use a (somewhat contrived) biological example, in which we wish to represent populations of wolves and deer which have different probabilities of survival depending on their location. A larger example can be found in [4]. To begin, we can introduce the concept of probabilities using the refinement:

$$\top \supseteq P : \mathit{probability}(P) \tag{10}$$

We could then go on to provide more specific information pertaining to probabilities. In particular, we could say that a more restricted type of *probability* is the survival factor of animals:

$$P : \mathit{probability}(P) \supseteq S : \mathit{survival}(A : \mathit{animal}(A), S) \tag{11}$$

At his point, we have introduced, as part of expression 11 a requirement for $\mathit{animal}/1$ to be placed in the lattice. This is flagged as one of the gaps in the requirement specification and we plug this gap by adding $\mathit{animal}/1$ below $\top$. At the same time, it is convenient to add $\mathit{wolf}/1$ and $\mathit{deer}/1$, as refinements of $\mathit{animal}/1$, and $\mathit{red_deer}/1$ as a refinement of $\mathit{deer}/1$:

$$\top \supseteq A : \mathit{animal}(A) \tag{12}$$

$$A : \mathit{animal}(A) \supseteq W : \mathit{wolf}(W) \tag{13}$$

$$A : \mathit{animal}(A) \supseteq D : \mathit{deer}(D) \tag{14}$$

$$D : \mathit{deer}(D) \supseteq R : \mathit{red_deer}(R) \tag{15}$$

We might then decide to introduce a refinement of *survival* which is dependent on the the location of the animals:

$$S : \mathit{survival}(A : \mathit{animal}(A), S) \supseteq F : \mathit{fl}(L : \mathit{location}(A : \mathit{animal}(A), L), B : \mathit{animal}(B), F) \tag{16}$$

This again introduces a gap in the specification, for $\mathit{location}/2$, which we first introduce below $\top$ and then define using two axioms:

$$\top \supseteq L : \mathit{location}(A : \mathit{animal}(A), L) \tag{17}$$

$$L : \mathit{location}(A : \mathit{animal}(A), L) \supseteq H : \mathit{hill}(H) \tag{18}$$

$$L : \mathit{location}(A : \mathit{animal}(A), L) \supseteq P : \mathit{pasture}(P) \tag{19}$$

We might now decide to be more specific about the types of results which we would expect to obtain from $\mathit{fl}/3$. For example, we could stipulate that the results in the third argument for deer on hills might be the integers between 50 and 100, while the same argument for wolves on hills might be the integers between 40 and 60.

$$F : fl(L : hill(L), A : deer(A), F) \supseteq N : between(50, 100, N) \quad (20)$$
$$F : fl(L : hill(L), A : wolf(A), F) \supseteq N : between(40, 60, N) \quad (21)$$

Finally, we could be more specific about the locations of particular groups of animals. For example, we could give possible locations for *red_deer* to be hills.

$$L : location(A : red_deer(A), L) \supseteq H : hill(H) \quad (22)$$

6 Extracting a Program

In Section 5 we demonstrated how a lattice of refinements could be constructed. This lattice is capable of describing a large number of different programs, which vary on two dimensions:

- The level of detail at which a program in the lattice is described will vary depending on the depth to which we descend through the chains of refinement. The further we travel towards the bottom of the lattice the more detailed our programs become.
- There may be more than one possible refinement of a set expression at any given point in the lattice. These produce choice points in the extraction of program details.

Bearing the above considerations in mind, the method used to extract a program from the refinement lattice is based on a simple principle. Recall the mapping between refinements and implication which has been shown in Section 3. Using this mapping, if we take any sequence of refinements down through the lattice from some top level set expression then by translating the refinements of that sequence into axioms of Prolog we shall have produced a partial program the results of which are included in the top–level set expression. Some additional complexity is introduced into the algorithm because we permit nesting of set expressions. This means that when we are finding sequences of refinements we need to do more than simply match the left and right sides of the appropriate refinements – we also need to ensure that set expressions contained in the matching expressions can be coerced toward a non–empty intersection. The full refinement algorithm is given below. Note the recursive use of the algorithm when unifying set expressions and also the need to propagate the set intersections from unification through the right hand side of the smaller of the refinement expressions.

Algorithm 1 *We write* $refinement(T_1, T_2, P)$ *to denote that* T_2 *is a valid refinement of* T_1 *producing axiom set* P*, given refinement lattice* $\mathcal{H}$*. The algorithm for this is as follows:*

- $refinement(T_1, T_2, P)$ *if* $refinement(T_1, T_2, \{\}, P)$

- $refinement(T, T, P, P)$
- $refinement(V : A, V : B, P, P'')$ *if*
  - $(V : A' \supseteq V : B') \in \mathcal{H}$ *and*
  - $unify(A, A', A_u, P, P')$ *and*
  - $propagate_bindings(A_u, B', B_u)$ *and*
  - $refinement(V : B_u, V : B, P', P'')$
- $refinement(P(A_1, \cdots, A_n), P(A'_1, \cdots, A'_n), P, P')$ *if*
  - *For each* A_I *and* A'_I: $refinement(A_I, A'_I, P_I)$ *and* $P' = \bigcup P_I \cup P$
- $refinement(V : A, V : B_1 \cap B_2, P, P'')$ *if*
  - $refinement(V : A, V : B_1, P, P')$ *and* $refinement(V : A, V : B_2, P', P'')$

Algorithm 2 *We write* $unify(A, A', A_u, P)$ *to denote that set expressions* A *and* A' *have a shared subset defined by set expression* A_u*, yielding axiom set,* P*. The algorithm for this is as follows:*

- $unify(A, A', A_u, P)$ *if* $unify(A, A', A_u, \{\}, P)$
- $unify(A, A', A_u, P, P'')$ *if*
  - $refinement(A, A_u, P, P')$ *and* $refinement(A', A_u, P', P'')$

Algorithm 3 *The procedure,* $propagate_bindings(A, B, B')$ *takes each term of the form* $V : X$ *contained in* A *and replaces every occurrence of* $V : _$ *in* B *with* $V : X$*, yielding the new term,* B'*.*

Using this refinement algorithm, in combination with axioms 10 to 22 from Section 5.1, we can extract a variety of sets of refinements from the refinement lattice, including the one below:

$$
\begin{aligned}
D : deer(D) &\supseteq D : red_deer(D) \\
D : animal(D) &\supseteq D : deer(D) \\
L : location(D : red_deer(D), L) &\supseteq L : hill(L) \\
S : fl(L : hill(L), D : red_deer(D), S) &\supseteq S : between(50, 100, S) \\
S : survival(D : red_deer(D), S) &\supseteq S : fl(L : location(D : red_deer(D), L), \\
&\qquad D : red_deer(D), S)
\end{aligned}
$$

Applying the translation algorithm to these refinements gives the partial program:

$$
\begin{aligned}
deer(D) &\leftarrow red_deer(D) \\
animal(D) &\leftarrow deer(D) \\
location(D, L) &\leftarrow red_deer(D) \;\&\; hill(L) \\
fl(L, D, S) &\leftarrow hill(L) \;\&\; red_deer(D) \;\&\; between(50, 100, S) \\
survival(D, S) &\leftarrow red_deer(D) \;\&\; location(D, L) \;\&\; fl(L, D, S)
\end{aligned}
$$

7 Conclusions

The language introduced in this paper embodies what we claim to be a novel approach to requirements capture. It has the following features:

- The space of requirements is described using a lattice of refinements between sets of potential results from Prolog programs.
- Construction of a Prolog program can be achieved by searching this requirement space, having delimited the upper and lower bounds within which the completed (partial) program must lie.
- Guidance during the construction of the refinement lattice is obtained by the application of logically consistent set–theoretic proof rules.

References

[1] A. Bundy and M.. Uschold. The use of typed lambda calculus for requirements capture in the domain of ecological modelling. Research Paper 446, Dept. of Artificial Intelligence, Edinburgh, 1989.

[2] J. Levy, J. Agusti, F. Esteva, and P. Garcia. An ideal model of an extended lambda-calculus with refinement. Ecs-lfcs-91-188, Laboratory for the Foundations of Computer Science, 1991.

[3] D. McAllester, B. Givan, and T. Fatima. Taxonomic syntax for first order inference. In *Proceedings of KR-89*, 1989.

[4] D. Robertson, J. Agusti, J. Hesketh, and J. Levy. Expressing program requirements using refinement lattices. Research paper, Department of Artificial Intelligence, University of Edinburgh, 1992. Longer version of paper submitted to ISMIS-93.

[5] D. Robertson, A. Bundy, R. Muetzelfeldt, M. Haggith, and M Uschold. *Eco-Logic: Logic-Based Approaches to Ecological Modelling.* MIT Press (Logic Programming Series), 1991. ISBN 0-262-18143-6.

[6] D. Robertson, M. Uschold, A. Bundy, and R. Muetzelfeldt. The eco program construction system: Ways of increasing its representational power and their effects on the user interface. *International Journal of Man Machine Studies*, 31:1–26, 1988.

Finding Logical Consequences Using Unskolemization

Ritu Chadha and David A. Plaisted
Department of Computer Science, University of North Carolina
Chapel Hill, N. C. 27599-3175.*

Abstract : This paper presents a method for deriving logical consequences of first-order formulas based on resolution and a novel unskolemization algorithm. In general, it is not possible to derive certain logical consequences of a first-order formula by resolution without using tautologies or unskolemization. Therefore, if a formula H implies a formula W, we will not attempt to derive W from H; instead, we derive a formula F such that H implies F and F implies W, and such that F is "close" to W. A measure of closeness is defined such that the number of formulas F "close" to any given formula W is finite. A number of interesting applications are discussed, including a method for mechanically generating loop invariants for program verification, and a technique for learning characteristic descriptions of objects.

1. Objective and Motivation

In this paper, we will develop a method for finding logical consequences of first-order formulas. Suppose we are given a first-order formula H, and we want to find a certain consequence W of H, which is unknown. It may not be possible to derive W from H by resolution [9] without using tautologies and unskolemization, as will be shown in Section 2.1. Since the use of tautologies is undesirable (due to the enormous increase in search space that it creates), we will not attempt to derive W from H, but instead will try to derive a formula F with the property that

$$H \Rightarrow F \Rightarrow W.$$

(where $\Rightarrow$ denotes logical implication). However, if this is the only constraint on F, then why not take F = H? One obvious reason is that H may be infinite. Also, we want F to be as "close" as possible to W. To define the concept of "closeness", we will define a relation "more general than" on first-order formulas and will derive a formula F from H such that $H \Rightarrow F \Rightarrow W$ and such that F is "more general than" W. The relation "more general than" is defined in such a way that the number of first-order formulas F which satisfy a given syntactic condition and are more general than a given first-order formula W is finite up to variants. Thus we can only derive a finite number of formulas F satisfying both the following conditions:

(i) $H \Rightarrow F \Rightarrow W$

(ii) F is more general than W.

Of course, if H is more general than W, then we could get F=H. We will show that this method is complete, i.e. that for any two formulas H and W, it is possible to derive F from H by our method such that (i) and (ii) above hold.

This paper is structured as follows. In the next section, we show why certain logical consequences of first-order formulas cannot be derived without using tautologies or unskolemization, and describe an unskolemization algorithm. The algorithm is analyzed in Section 3, where we define the relation "more

* Ritu Chadha may be contacted at the following address : Bell Communications Research, MRE 2A-246, 445 South Street, Morristown, NJ 07962-1910.

general than". Several applications for the methods developed in this paper are described in Section 4. Section 5 concludes with a brief summary.

2. The Unskolemization Process

2.1 Preliminaries

Unskolemization has been defined as the process of eliminating Skolem functions from a formula without quantifiers, replacing them with new existentially quantified variables, and transforming the resulting formula into a closed formula with quantifiers (for details about skolemization, see [2,7]). McCune [8] presents an algorithm to solve the following problem : given a set S of clauses and a set F of constant and function symbols that occur in the clauses of S, obtain a fully quantified (closed) formula S' from S by replacing expressions starting with symbols in F with existentially quantified variables. McCune's algorithm is sound but not complete. Cox and Pietrzykowski [3] present an algorithm for unskolemization, but their algorithm is applicable only to literals.

We expand the meaning of unskolemization slightly. In our definition, function symbols can also be "unskolemized" by treating them as if they were Skolem functions. Thus, a function symbol may be replaced by an existentially quantified variable during unskolemization. To illustrate, suppose we want to unskolemize the formula $\forall x(P(f(x)) \vee Q(g(a), x))$ where f and a are (non-Skolem) function symbols, and suppose we want to treat f and a as if they were Skolem functions. The resulting formula would be $\exists z \forall x \exists y(P(y) \vee Q(g(z), x))$. Note that skolemizing $\exists z \forall x \exists y(P(y) \vee Q(g(z), x))$ yields the original formula (up to names of Skolem functions). In practice, the situation may be more complicated, since the formula being unskolemized may not be the skolemized form of any formula. Our algorithm shows how to cope with such situations.

We motivate the development of the unskolemization algorithm by the following example. Suppose we want to derive an unknown logical consequence B of A. Denote the Skolem form of a formula F by "Sk(F)". Since $A \Rightarrow B$, $A \wedge \neg B$ is unsatisfiable, so Sk($A \wedge \neg B$) is unsatisfiable (since skolemization preserves unsatisfiability), i.e. Sk(A)$\wedge$ Sk($\neg B$) is unsatisfiable. Therefore we can derive the empty clause from Sk(A)$\wedge$ Sk($\neg B$). Now, B is unknown, and we want to derive it from Sk(A). It may not be possible to derive B from Sk(A) without using tautologies or unskolemization, as demonstrated below:

(i) Suppose $A = P$ and $B = P \vee Q \vee R$. Clearly $A \Rightarrow B$. But the only way to derive B from A by resolution is by resolving A with the tautology $\neg P \vee P \vee Q \vee R$.

(ii) Suppose $A = P(a)$ and $B = \exists x P(x)$. Then B (or even Sk(B)) cannot be derived from A by resolution. Obtaining B from A requires unskolemizing A by replacing "a" by an existentially quantified variable. Unskolemizing $P(a)$ results in $\exists x P(x)$. In practice, some function symbols in A may have to be replaced by existential quantifiers and some may not. This explains why our unskolemization algorithm will be nondeterministic.

In order to address the above issues formally, we present an unskolemization algorithm U with the following specifications:

INPUT: a first-order formula H
OUTPUT: set $\mathcal{L}$ of formulas such that for any logical consequence W of H, algorithm U can produce a formula **F** in $\mathcal{L}$ such that

(i) H $\Rightarrow$ **F** $\Rightarrow$ W
(ii) **F** is more general than W

where "more general than" is a relation that will be defined later with the property that $\{F \mid F$ is more general than $W\}$ is finite upto variants under certain syntactic constraints. ■

The algorithm U unskolemizes a set of clauses $\mathcal{D}$ derived by resolution from Sk(H) to give a set of formulas $\mathcal{L}$. Briefly, the objective of unskolemizing $\mathcal{D}$ is to replace function symbols of $\mathcal{D}$ that do not occur in W by existentially quantified variables. That is, if for some literal L in $\mathcal{D}$, an argument d of L has a function symbol that does not appear in W, then that function symbol of d is unskolemized, yielding a set $\mathcal{L}$ of new formulas. Thus any $F \in \mathcal{L}$ will contain a new existentially quantified variable in place of d. Since W is unknown, this procedure will have to be carried out nondeterministically. This process will make the unskolemized formula "more general than" W.

Notes. 1. The following algorithm makes use of the guarded command for conditional statements [5]. Briefly, the general form of a conditional statement is "**if** $B_1 \rightarrow S_1$ [] $B_2 \rightarrow S_2$ ... [] $B_n \rightarrow S_n$ **fi**", where $n \geq 0$ and each $B_i \rightarrow S_i$ is a guarded command. Each S_i can be any statement. The command is executed as follows. If any guard B_i is not well-defined, or if none of the guards is true, abortion occurs; if at least one guard is true, then one guarded command $B_i \rightarrow S_i$ with true guard B_i is chosen and S_i is executed. If more than one guard is true, then one of the guarded commands $B_i \rightarrow S_i$ with true guard B_i is chosen arbitrarily and S_i is executed. Thus the execution of such a statement can be nondeterministic.
2. The following notation is used:
(i) L = SIGN(L) $P(a_1, ..., a_n)$ is a literal whose sign (negated or unnegated) is represented by "SIGN(L)"; e.g. SIGN($Q(a)$)=" ", SIGN($\neg Q(a)$)="$\neg$".
(ii) Let X be a term. FUNC(X) is defined to be the function symbol of X if X is not a variable, and is defined to be X otherwise. For example, FUNC($f(x, y)$) = f; FUNC(a) = a; FUNC(x) = x, where x is a variable.

2.2 The Unskolemization Algorithm

ALGORITHM U

Step 1. Skolemize the input formula H. Let SK be the set of all the Skolem symbols in Sk(H). Derive a set $\mathcal{D}$ of clauses by resolution from Sk(H).

Step 2. Make i_k copies of every clause C_k of $\mathcal{D}$, where i_k is some integer (chosen nondeterministically), and rename variables in all clauses so that no two clauses have any variable in common. Call the resulting bag of clauses M_CLAUSES.
Comment : We may need multiple copies of clauses because multiple instances of a clause may be needed to derive the empty clause from Sk($H \wedge \neg W$). i_k can be bounded by the number of resolutions performed when deriving the empty clause from Sk($H \wedge \neg W$). In actual practice, for each k, we can try setting i_k to 1, then 2, and so on, and eventually i_k will become large enough.

Step 3. For every literal L in every clause of M_CLAUSES, process the arguments of L as follows. Suppose L=SIGN(L) $P(d_1, d_2, ..., d_s)$ For each i, $1 \leq i \leq s$, perform the following:

if (FUNC(d_i)$\in$SK) $\rightarrow$ replace d_i by X $\leftarrow d_i$, for some fresh variable X;
[] (FUNC(d_i)$\notin$SK $\wedge$ d_i is not a variable) $\rightarrow$
    replace d_i by X $\leftarrow d_i$, for some fresh variable X;
[] (FUNC(d_i)$\notin$SK $\wedge$ d_i is not a variable) $\rightarrow$ skip;
[] (d_i is a variable) $\rightarrow$ skip

fi

Call the resulting set of processed clauses MARK.

Comment : Replacing d_i by $X \leftarrow d_i$ is just a way of marking d_i with a variable name. Any argument of the form "$X \leftarrow d_i$" is called a *marked* argument.

Step 4. For every pair of marked arguments "$X \leftarrow \alpha$", "$Y \leftarrow \beta$" in MARK do
if α, β are unifiable $\rightarrow$ unify all occurrences of X and Y;
[] α, β are unifiable $\rightarrow$ skip;
[] α, β are not unifiable $\rightarrow$ skip
fi

Comment : In the next step, **C** is the set of constraints on the ordering of new existential quantifiers relative to universal quantifiers which will be introduced in Step 6. The presence of an ordered pair (y, z) in **C** signifies that "$\exists z$" must come *after* "$\forall y$" in the quantifier string of the unskolemized formula.

Step 5. Let **C** be a set which is initially empty, and let Q be an initially empty quantifier string. Let FREE be the set of all free variables in MARK (this does not include marked arguments). For every marked argument "$x \leftarrow \alpha$" do

{Collect all marked arguments with the same variable on the left-hand side of the "$\leftarrow$" sign. Suppose these are

$$x \leftarrow \alpha_1, x \leftarrow \alpha_2, ..., x \leftarrow \alpha_n.$$

Let $\{y_1, y_2, ..., y_r\}$ be the set of all the variables occurring in $\alpha_1, \alpha_2, ..., \alpha_n$. Then replace "$x \leftarrow \alpha_i$", for $1 \leq i \leq n$, everywhere by a new variable z (say) and add the r ordered pairs (y_i, z) to **C**. If $r = 0$, place "$\exists z$" at the head of the partially completed quantifier string Q.}

Step 6. (i) For every y in FREE, define $\mathrm{DEP}(y) = \{z | (y, z) \in \mathbf{C}\}$. This is the set of all variables z such that "$\exists z$" must come after "$\forall y$". Define the partial order PO on the set FREE as follows: $(x, y) \in$ PO iff $\mathrm{DEP}(x) \supseteq \mathrm{DEP}(y)$.
(ii) Extend the partial order PO to a linear order on FREE in all possible ways, yielding a set LIN of linear orders.
(iii) Let QUANT be an initially empty set of quantifier strings. For every linear order O in LIN do

{P := Q;
add universal quantifiers for every variable in FREE to P in the order prescribed by O (i.e. if $x < y$ in O, then $\forall x$ precedes $\forall y$ in P). These quantifiers come *after* any existential quantifiers already present in Q;
QUANT := QUANT $\cup$ {P} }

(iv) Let $\mathcal{L}$ be an initially empty set. For every Q in QUANT do

{Insert an existential quantifier for every z such that $(y, z) \in$ **C** (for some y) as far forward in Q as possible, subject to the constraint that "$\exists z$" comes after "$\forall y$" for every y such that $(y, z) \in$ **C**;
Rewrite MARK in conjunctive normal form;
Add the formula "Q MARK" to the set $\mathcal{L}$. } ■

Example 1. Let

$H = \forall x \forall y \forall z \forall w \forall t((Q(y) \vee L(b, y, t)) \wedge \neg Q(g(t)) \wedge L(g(t), a, t) \wedge (R(x, g(t)) \vee \neg P(x, g(t))) \wedge (\neg R(w, z) \vee \neg D(w, z)))$,

$W = \forall s \exists u \forall v(L(b, u, s) \wedge L(u, a, s) \wedge (\neg P(v, u) \vee \neg D(v, u) \vee M(a)))$.

It is easy to see that $H \Rightarrow W$. We will show how algorithm U derives a formula F from H such that $H \Rightarrow F \Rightarrow W$. We have,

$\neg W = \exists s \forall u \exists v((\neg L(b, u, s) \vee \neg L(u, a, s) \vee P(v, u)) \wedge (\neg L(b, u, s) \vee \neg L(u, a, s) \vee D(v, u)) \wedge (\neg L(b, u, s) \vee \neg L(u, a, s) \vee \neg M(a)))$

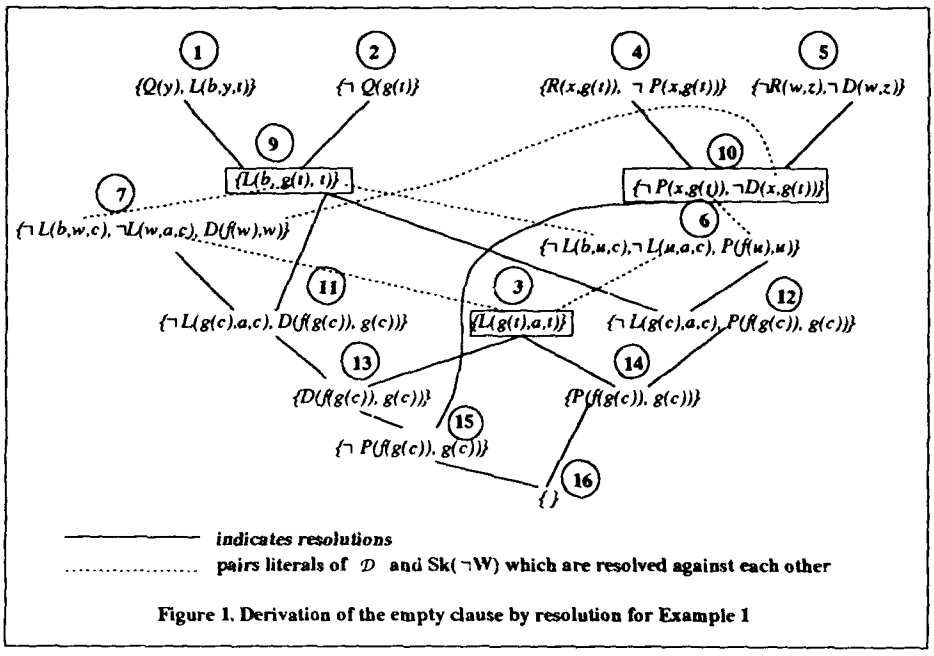


Figure 1. Derivation of the empty clause by resolution for Example 1

$\mathrm{Sk}(\neg W) = \{\{\neg L(b,u,c), \neg L(u,a,c), P(f(u),u)\}, \{\neg L(b,u,c), \neg L(u,a,c), D(f(u),u)\}, \{\neg L(b,u,c), \neg L(u,a,c), \neg M(a)\}\}$.

In $\mathrm{Sk}(\neg W)$, f and c are Skolem functions replacing the existentially quantified variables v and s of $\neg W$, respectively. A sequence of resolutions that derives the empty clause from $\mathrm{Sk}(H) \wedge \mathrm{Sk}(\neg W)$ is depicted pictorially in Figure 1. Notice that resolutions among clauses of $\mathrm{Sk}(H)$ were performed first and then some of these clauses were used during the remainder of the resolution process. We will define the set $\mathcal{D} \subset \mathrm{Sk}(H)$ to consist of the three clauses $\{\{L(b,g(t),t)\}, \{L(g(t),a,t)\}, \{\neg P(x,g(t)), \neg D(x,g(t))\}\}$ which were obtained from $\mathrm{Sk}(H)$ and were used to derive the empty clause from $\mathrm{Sk}(H) \wedge \mathrm{Sk}(\neg W)$. These clauses are enclosed in boxes in Figure 1.

INPUT : The formula H given above.

Step 1: We define the set $\mathcal{D}$ to consist of the following three clauses, as explained above: $\{\{L(b,g(t),t)\}, \{L(g(t),a,t)\}, \{\neg P(x,g(t)), \neg D(x,g(t))\}\}$.

Step 2: Since the clauses $\{L(b,g(t),t)\}$ and $\{L(g(t),a,t)\}$ are both used twice during the resolution in Figure 1, we make two copies each of these clauses, and make one copy of the clause $\{\neg P(x,g(t)), \neg D(x,g(t))\}$. We now have the bag of clauses M_CLAUSES consisting of the following five clauses: $\{\{L(b,g(t),t)\}, \{L(b,g(t),t)\}, \{L(g(t),a,t)\}, \{L(g(t),a,t)\}, \{\neg P(x,g(t)), \neg D(x,g(t))\}\}$.

Step 3: Now "mark" arguments of M_CLAUSES as follows. Look at which literal of $\mathrm{Sk}(\neg W)$ each of the above literals resolves against in Figure 1. Pairs of literals of $\mathcal{D}$ and $\mathrm{Sk}(\neg W)$ that resolve against each other are linked by dotted lines in the figure. We see that $L(b,g(t),t)$ resolves against $\neg L(b,w,c)$; $L(b,g(t),t)$ resolves against $\neg L(b,u,c)$; $L(g(t),a,t)$ resolves against $\neg L(w,a,c)$; $L(g(t),a,t)$ resolves against $\neg L(u,a,c)$; $\neg P(x,g(t))$ resolves against $P(f(u),u)$; and $\neg D(x,g(t))$ resolves against $D(f(w),w)$.

Note: By looking at (for instance) the resolution between clauses 3 and 11, which yields clause 13, it appears that $L(g(t), a, t)$ (from clause 3) resolves against $\neg L(g(c), a, c)$ (from clause 11). However, the literal $\neg L(g(c), a, c)$ in clause 11 is an instance of the literal $\neg L(w, a, c)$ in clause 7. Clause 7 belongs to Sk($\neg W$) (and clause 11 doesn't). Thus the literal in Sk($\neg W$) that $L(g(t), a, t)$ resolves against is $\neg L(w, a, c)$ in this case.

For any function symbol F in a literal of M_CLAUSES which resolves against a variable X in Sk($\neg W$), mark it by replacing F by "X $\leftarrow$ F". This yields the following set of marked clauses MARK: $\{\{L(b, w \leftarrow g(t), t)\}, \{L(b, u \leftarrow g(t), t)\}, \{L(w \leftarrow g(t), a, t)\}, \{L(u \leftarrow g(t), a, t)\}, \{\neg P(x, u \leftarrow g(t)), \neg D(x, w \leftarrow g(t))\}\}$.

Step 4: We unify some of the variables on the left hand side of the "$\leftarrow$" in marked arguments. To decide which variables will be unified, we look at variables in unmarked arguments of MARK. There are two such variables, namely x and t. These variables were unified with $\{f(u), f(w)\}$ and $\{c\}$ respectively (see the preceding analysis). Since x was unified with both $f(u)$ and $f(w)$, we unify $f(u)$ with $f(w)$. MARK now contains the five clauses $\{L(b, u \leftarrow g(t), t)\}$, $\{L(b, u \leftarrow g(t), t)\}$, $\{L(u \leftarrow g(t), a, t)\}$, $\{L(u \leftarrow g(t), a, t)\}$, $\{\neg P(x, u \leftarrow g(t)), \neg D(x, u \leftarrow g(t))\}$. Since the first and second clauses are identical, and so are the third and fourth clauses, we can drop the duplicate clauses. MARK now consists of the three clauses : $\{\{L(b, u \leftarrow g(t), t)\}, \{L(u \leftarrow g(t), a, t)\}, \{\neg P(x, u \leftarrow g(t)), \neg D(x, u \leftarrow g(t))\}\}$.

Step 5: Here FREE = $\{t, x\}$. The marked arguments in MARK are "$u \leftarrow g(t)$". We replace these arguments by a fresh variable Z, and add the pair (t, Z) to **C**. This yields **C** = $\{(t, Z)\}$ and MARK consists of the clauses $\{\{L(b, Z, t)\}, \{L(Z, a, t)\}, \{\neg P(x, Z), \neg D(x, Z)\}\}$.

Step 6: (i) Here DEP(t) = $\{Z\}$, DEP(x) = $\{\ \}$.
Since DEP(t) $\supseteq$ DEP(x), PO = $\{(t, x)\}$.
(ii) The partial order PO is a linear order on FREE; thus LIN = {PO}.
(iii) QUANT = $\{\forall t \forall x\}$
(iv) The existential quantifier for Z must be placed after $\forall t$, as far forward as possible; thus QUANT = $\{\forall t \exists Z \forall x\}$, and the resulting set of formulas is

$$\mathcal{L} = \{\forall t \exists Z \forall x (L(b, Z, t) \wedge L(Z, a, t) \wedge (\neg P(x, Z) \vee \neg D(x, Z)))\}.$$

Thus there is only one formula in the set $\mathcal{L}$ for this example; call it **F**. It can easily be verified that H $\Rightarrow$ **F** $\Rightarrow$ W. ■

3. Analysis of the Unskolemization Algorithm

This section defines the "more general than" relation and lists some theorems which prove that algorithm U satisfies its specification. Proofs for the theorems in this section can be found in [1].

Theorem 1. Let H, W be formulas such that $H \Rightarrow W$. If the right non deterministic choices are made, given H as input, algorithm U produces a set $\mathcal{L}$ of formulas such that for any formula F in $\mathcal{L}$, for any literal L = SIGN(L) $P(d_1, d_2, ..., d_s)$ of Sk(F), there exists a literal M of W such that M = SIGN(M) $P(b_1, b_2, ..., b_s)$, SIGN(M) = SIGN(L), and such that for all i, $1 \leq i \leq s$,

(i) If d_i is a *Skolem function*, then b_i is existentially quantified in W.

(ii) If d_i is a non-Skolem function, then one of the following holds:

(a) b_i is the same function, and (i) and (ii) here hold recursively for each corresponding argument of d_i and b_i.

(b) b_i is existentially quantified and the function symbol of d_i (with the same arity as d_i) appears in W. ■

Intuitively, we are trying to say that Skolem symbols in $\mathcal{D}$ (where $\mathcal{D}$ is as specified in Step 1 of algorithm U) are replaced by existentially quantified variables in W ((i) in the theorem statement), and non-Skolem function symbols in $\mathcal{D}$ *which do not appear* in W are also replaced by existentially quantified variables. Thus any non-Skolem function symbol that remains in an unskolemized formula F must appear somewhere in W ((ii) in the theorem statement). This is crucial because it allows us to define a relation "more general than" such that the number of formulas more general than a given formula is finite under certain syntactic constraints. Also, the fact that every literal L in Sk(F) has a corresponding literal M in W with the same predicate, arity, and sign shows that formula F is similar to W in the predicates that it contains.

Motivated by Theorem 1, we introduce the following definition.

Definition. A formula F is *more general than* a formula W if for every literal L of F, there exists a literal M of W such that if L = SIGN(L)$P(a_1, a_2, ..., a_s)$, then M = SIGN(M)$P(b_1, b_2, ..., b_s)$, where SIGN(L) = SIGN(M), and for all i such that $1 \leq i \leq s$,

(i) If a_i is an existentially quantified variable, then so is b_i.

(ii) If a_i is a function symbol followed by u arguments $e_1, e_2, ..., e_u$, then

(a) b_i is the same function symbol followed by the same number of arguments, say $f_1, f_2, ..., f_u$, and conditions (i) and (ii) hold for every pair of arguments e_k and f_k, $1 \leq k \leq u$, or

(b) b_i is an existentially quantified variable and a_i has a function symbol that occurs in W. ■

Note the similarity between the statement of Theorem 1 and this definition. The definition of "more general than" given above does not allow function symbols that do not appear in W to appear in F if F is more general than W.

Corollary to Theorem 1. For every $F \in \mathcal{L}$, F is more general than W.

Definition. Let F, W be two first-order formulas. We say that $F \preceq W$ iff

(i) F is more general than W

(ii) $F \Rightarrow W$.

Example 3. We illustrate the meaning of "more general than".

(i) $F = \forall x \exists y \forall z (P(x, y) \wedge Q(y, z))$, $W = \forall u \exists v P(u, v)$.

F is not more general than W because for the literal $Q(y, z)$ in F, there is no literal in W with the specified properties, since W does not even have a literal with predicate symbol Q.

(ii) $F = \forall x \exists y \forall z (P(x, y) \wedge Q(y, z))$, $W = \forall u \exists v (P(u, v) \wedge Q(v, v))$.

Here F is more general than W, because for $P(x, y)$ in F, there is a corresponding literal $P(u, v)$ in W with the specified properties; similarly, for $Q(y, z)$ in F, there is a corresponding literal $Q(v, v)$ in W with the specified properties.

Also, $F \Rightarrow W$; therefore $F \preceq W$.

(iii) $F = \forall x \exists y \forall z (P(x, y) \vee Q(y, z))$, $W = \forall u \exists v (P(u, v) \wedge Q(v, v))$.

As in (ii), F is more general than W. However, $F \not\Rightarrow W$; therefore $F \not\preceq W$. ■

Theorem 2. $\{F | F \preceq W\}$ is finite up to variants, assuming that if F is written in conjunctive normal form, then no two disjunctions of F are identical, and no disjunction of F contains more than one occurrence of the same literal. ■

Theorem 3. For every $F \in \mathcal{L}$, $H \Rightarrow F$, where $\mathcal{L}$ is the set of formulas obtained by unskolemizing a formula H according to algorithm U. ■

Theorem 4. Given formulas H, W such that $H \Rightarrow W$, there exists $\mathbf{F} \in \mathcal{L}$ such that $\mathbf{F} \Rightarrow W$, where H and $\mathcal{L}$ are the input and output of algorithm U respectively. ■

Corollary to Theorem 4: There exists $\mathbf{F} \in \mathcal{L}$ such that $\mathbf{F} \preceq \mathrm{W}$. ■

4. Applications

The unskolemization algorithm discussed in this paper was originally developed as part of a mechanism for deriving loop invariants for program loops, with a view to automatically proving the partial correctness of programs. In Floyd's inductive assertions method for proving the partial correctness of programs [4], the user is required to supply loop invariants for every loop in the program. Although some attempts have been made in the past to mechanize the derivation of loop invariants, most of the methods developed are heuristic in nature (for two basic heuristic approaches, see [6,10]). We can describe the problem of generating loop invariants as one of generating logical consequences of an infinite number of formulas. To see this, suppose that W is a loop invariant for a program loop at the entry to the loop. Let A_i be the formula which holds before the i^{th} iteration of the loop. Then $A_i \Rightarrow W$ for all i such that $i \geq 1$. We have, $A_1 \Rightarrow W$, $A_2 \Rightarrow W$, $A_3 \Rightarrow W$, ..., and so on, i.e. W is a logical consequence of each A_i, for all i such that $i \geq 1$. Thus the method described in this paper for deriving logical consequences can be used for deriving W.

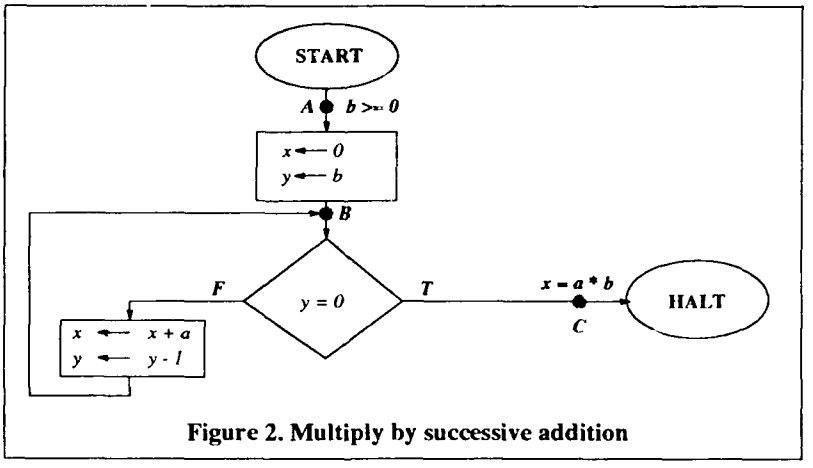


Figure 2. Multiply by successive addition

As a quick and simple example, consider the program in Figure 2, which multiplies two numbers by successive addition. Suppose we are trying to find a loop invariant for the loop of this program, which would always be true at point B. The input and output assertions for this program are attached at points A and C respectively. Using the preceding terminology, if W is a loop invariant for the program at point B, then the formulas $A_1, A_2, \ldots$, are:

$$A_1 \equiv b \geq 0 \wedge x = 0 \wedge y = b$$
$$A_2 \equiv b \geq 0 \wedge x = a \wedge y = b - 1$$
$$A_3 \equiv b \geq 0 \wedge x = 2 * a \wedge y = b - 2$$
$$A_4 \equiv b \geq 0 \wedge x = 3 * a \wedge y = b - 3$$

and so on. Let us unskolemize A_4 by replacing the (non-Skolem) symbol "3" by an existentially quantified variable. We get

$$\exists m(b \geq 0 \wedge x = m * a \wedge y = b - m)$$

which is a loop invariant for this loop which can be used to prove the partial correctness of this program. We hope that the simplicity of this example does not obscure its intent, which is to illustrate the need for unskolemization in situations like the above. The algorithm which we have developed for deriving loop invariants provides guidance to the unskolemization algorithm for deciding which symbols should be unskolemized. A complete description of this algorithm can be found in [1]. We have developed a complete algorithm for deriving loop invariants within the framework of first-order logic; in other words, if a loop invariant can be expressed in first-order logic, then our algorithm can generate a loop invariant, using resolution and unskolemization, which can be used to prove the partial correctness of the program. A proof of this is included in [1]. This proof rests on the crucial fact that the set $\{F|F \preceq W\}$ is finite as described in Theorem 2; thus if an algorithm for deriving loop invariants continually generates approximations F for a loop invariant W such that F $\preceq$ W, then since only a finite number of such F's exist, the algorithm converges and finds a suitable loop invariant.

The method for deriving logical consequences described here has also been applied to learning characteristic descriptions from examples. A characteristic description is a description of a collection of objects, situations or events which states facts that are true of all objects in the class. More formally, a statement S is a description of objects $O_1, O_2, O_3, \ldots$ if $O_1 \Rightarrow S, O_2 \Rightarrow S, O_3 \Rightarrow S, \ldots$ and so on. This provides a straightforward application of the method described in this paper, since the statement S to be derived is a logical consequence of $O_1, O_2, O_3, \ldots$. Another very simple example to illustrate the use of unskolemization is the following: consider the two sets of objects depicted in Figure 3. The objects in each set can be described in first-order logic as:

$SET_1 \equiv blank(a) \wedge small(a) \wedge square(a) \wedge blank(b) \wedge large(b) \wedge circle(b)$

$SET_2 \equiv blank(c) \wedge small(c) \wedge circle(c) \wedge shaded(d) \wedge large(d) \wedge circle(d)$

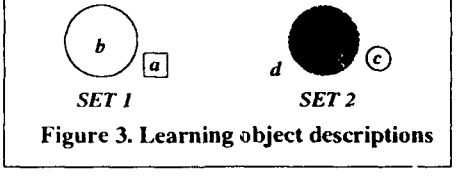


Figure 3. Learning object descriptions

Our learning algorithm then uses a maximum-matching algorithm for bipartite graphs to determine which clauses and symbols should be unskolemized. The resulting formula, in this case, is:

$$\exists x \exists y(blank(x) \wedge small(x) \wedge large(y) \wedge circle(y))$$

In other words, in each set of objects, there is a small blank object and a large circle. A description of a learning algorithm which makes use of the unskolemization algorithm is given in [1]. This learning algorithm builds on the work of this paper, since it provides a means of eliminating some of the nondeterminism of the unskolemization algorithm by finding structural similarities between examples.

Finally, we have applied our method for deriving logical consequences to

the mechanization of mathematical induction in first-order logic theorem provers. The principle of mathematical induction cannot be expressed in first-order logic; therefore, in order to enable first-order logic theorem provers to prove theorems requiring the use of the induction principle, it is necessary to provide the provers with the necessary inductive hypotheses which they will need in order to prove a theorem. These hypotheses can be derived as negations of logical consequences of AXIOMS $\wedge\neg$T, where AXIOMS is a set of axioms for the domain involved, and T is the theorem to be proved. Thus our method for deriving logical consequences can be used for this task. The method is described in [1].

5. Summary

This paper has presented a method for deriving logical consequences of first-order formulas based on resolution and unskolemization. A number of applications (such as those described in the previous section) require the generation of logical consequences of first-order formulas. It was shown that it is impossible to generate all possible logical consequences of a first-order formula without using tautologies or unskolemization. We therefore presented an unskolemization algorithm that, given a first-order formula H, can derive formulas F which are "close" to logical consequences W of H. A measure of closeness is defined using a "more general than" relation. The suitability of our measure is demonstrated by proving that only a finite number of formulas are "close" to any particular formula, by our definition. We would like to point out that although the algorithm contains some nondeterminism, much of this nondeterminism is removed when the algorithm is used for the applications described in Section 4. The nondeterminism of the algorithm gives it the generality to be taken and applied in a variety of different ways.

References

1. R. Chadha: *Applications of Unskolemization.* Ph.D. dissertation, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill (1991).
2. C. Chang, R.C. Lee: *Symbolic Logic and Mechanical Theorem Proving*, Academic Press Inc., New York (1973).
3. P.T. Cox, T. Pietrzykowski: A complete, nonredundant algorithm for reversed skolemization. *Theoretical Computer Science, 28*, 239-261 (1984).
4. R.W. Floyd: Assigning meanings to programs. *Proc. Symp. on Applied Mathematics*, American Mathematical Society, 19, 19-32 (1967).
5. D. Gries: *The Science of Programming.* Springer-Verlag (1981).
6. S.M. Katz, Z. Manna: A heuristic approach to program verification. *Proc. Third Intl. Joint Conf. on Artificial Intelligence*, 500-512 (1973).
7. D. Loveland: *Automated Theorem Proving , A Logical Basis.* North-Holland Publishing Co. (1978).
8. W.W. McCune: Un-Skolemizing clause sets. *Information Processing Letters*, 257-263 (1988).
9. J.A. Robinson: Machine-oriented Logic based on the Resolution Principle. *Journal of the ACM, 12* (1), 23-41 (1965).
10. B. Wegbreit: Heuristic Methods for Mechanically Deriving Inductive Assertions. *Proc. Third Intl. Joint Conf. on Artificial Intelligence* (1973).

Controlled Explanation Systems

Arcot Rajasekar

Department of Computer Science, University of Kentucky, Lexington, KY 40506

Abstract. In this paper we extend the definition of explanations to controlled explanations. Traditionally, given a set of facts and a theory, a set of explanations are generated which can be used to infer the facts. When the number of such explanations are more than one, then some criteria of minimality or preference is adapted to select some of the explanations. Most of these selection criteria are syntactic based and are domain-independent. In this paper, we define a system, where the selection can be made using some domain-dependent criteria. We motivate and define controlled explanations, show some of their properties and provide a procedure which generates minimal controlled explanations.

1 Introduction

Explanation systems form the heart of several artificial intelligence systems such as truth maintenance systems [3, 1], abductive reasoning systems [7], diagnostic systems [8, 2] and expert systems. The underlying theme of all these systems is to explain a set of facts using a subset of a given theory. In most cases, such as circuit-fault diagnosis, it is normal to use the explanation system to explain an abnormal set of observations in terms of defective (or abnormal) parts in the system. In some cases, such as abductive reasoning used in medical diagnosis, the aim is to find a set of diseases which can explain an observed set of symptoms. In some other cases, such as a calculus tutoring system, it is required to find all methods of solving a given problem. In either case, one normally is interested not just in every explanation possible, the number may be overwhelming, but only in a small set of explanations which have some preference property. There are two distinct methods used in selecting explanations. The first method of preference is based on some probabilistic criteria such as confidence factors. The second method of selection is based on set-theoretic criteria such as cardinality-minimal explanations (as in generalized set covering (GCS) explanations [7]) or set-minimal explanations (as in minimal diagnosis of [8]). Both these methods have their advantages and disadvantages: the probabilistic preference systems provide semantic selection, but they have problems with transitivity and in providing numeric confidence factors. The set-theoretic criteria are syntactic and do not use any semantic information for ordering their explanations; the user is left to choose among the final set.

In this paper, we are interested in semantic-based domain-dependent criteria which can be used in conjunction with set-theoretic criteria to reduce the number of explanations that are generated. This semantic information which controls the production of explanations is provided by the user in addition to the facts

that need to be explained. The semantic information is either about *what should be part of an explanation* or about *what cannot be part of an explanation*. The first type can be termed as *coercion* and the second as *restriction*. Both of them control the explanation generation process. The controlled explanation approach can be illustrated as follows: Consider that a doctor is examining a patient and finds the symptoms of headache and runny nose. He also finds that the patient does not have any chest congestion. Now there are a number of causes for headache and runny nose: common-cold, migraine, influenza, sinusitis, pneumonia, etc. Of these only migraine and sinus headaches do not have any chest congestion which can lead to a runny nose[1]. Hence, the doctor would factor in the fact that the patient does not have any chest congestion in his analysis to eliminate all other explanations, except for migraine and sinus. Logically, one can look at the situation as follows. There are three rules for runny noses (in our databases) and one of them is inapplicable. Hence any explanation based on that rule is bound to be wrong and should not be generated. Hence the data of no chest congestion is a *restriction* condition to reduce our set of explanations. Further, assume that the patient says that the headache is throbbing. This additional factor can be used by the doctor to prefer an explanation which uses this fact over others: possibly sinusitis over migraine. Note that the fact of the headache being throbbing is not an important one, i.e. not important enough to form part of the symptoms list, but if the explanation uses this fact then that explanation is all the more preferable. Logically, throbbing headache is a kind of headache. Just using throbbing headache instead of headache as part of the symptoms might shut out a lot of explanations, (for example, a headache due to influenza may or may not be throbbing, and using the symptom throbbing headache may shut out this diagnosis, even if it is supported by other symptoms). Hence using throbbing headache to prefer a particular diagnosis can be seen as the use of *coercion* to find a preferable diagnosis.

In this paper we develop a syntax and semantics for controlled explanations. We restrict our attention to the restriction type of control and provide a proof procedure for obtaining such explanations. The extension of the system to coercion type control is discussed elsewhere [6]. In the next section we develop the syntax and semantics for restriction-controlled explanations. In Section 3, we provide a proof procedure for obtaining such explanations. We conclude our paper with a discussion in Section 4.

2 Controlled Explanation - Definition and Properties

In this paper we develop a theory behind controlled explanation based on the logic programming framework. But the approach is general enough that it can be adapted for other types of explanation systems such as cause-effect systems [7], default theory based systems [5], and other diagnostic systems [8, 2].

[1] runny nose in migraine headache is probably caused by watering of the eyes. The same in sinus may be due to discharges.

We assume that the theory T which is the basis for generating explanations is made of normal Horn clauses [4] (with negation in the body, which are treated as a non-classical negation). We also assume that the reasoning mechanism uses first order inference axioms augmented with some meta-rule for treating negation (eg. the closed world assumption (or one its extended forms) may be an appropriate mechanism for treating negation.) The *consequences* of T under some reasoning model (mechanism) $\mathcal{R}$ is denoted as $Cn_{\mathcal{R}}(T)$. That is, $Cn_{\mathcal{R}}(T) = \{m : T \models_{\mathcal{R}} m\}$, where the mechanism $\models_{\mathcal{R}}$ depends upon the underlying theory and axiom schema being used. If $\mathcal{L}$ is propositional, then the axioms of propositional calculus provides one such mechanism In this paper, since we are concerned with theories which are made of Horn programs, the mechanism of $\models_{\mathcal{R}}$ is given by the declarative semantics of various classes of logic programs, such as the least model semantics for Horn programs, supported model for stratified programs, perfect model semantics for locally stratified programs, stable model semantics or well-founded semantics for normal logic programs. The definition of a controlled explanation is general enough to apply for many types of theories.

We also have a notion opposite to that of consequences; that of *non sequiturs*. We denote *non sequiturs* of T under $\mathcal{R}$ by $Ns_{\mathcal{R}}(T)$. That is, $Ns_{\mathcal{R}}(T) = \{m : T \not\models_{\mathcal{R}} m\}$. The twin concepts of consequences and non-sequiturs provide a basis for developing a theory for explanations. We first require the notion of an inference pair. An inference pair is a two-tuple, $< s, u >$, where s (resp. u) is a subset of $\mathcal{L}$ and is called the consequence part (resp. non-sequitur part) that needs to be explained. Any explanation S for the inference pair $< s, u >$ is a subset of a larger theory T such that s is in $Cn_{\mathcal{R}}(S)$ and u is in $Ns_{\mathcal{R}}(S)$. In the case of normal Horn programs, $Cn_{\mathcal{R}}(T)$ consists of a sets of atoms and $Ns_{\mathcal{R}}(T)$ consist of set of atoms and negated atoms. Whenever one needs to explain a phenomenon or observation (say p), then p should be in the consequence part of the inference pair and whenever one needs the explanation to refrain from using another phenomenon (say q) in the explanation, then q should be in the non-sequitur part of the inference pair. For example, the patient with the headache example of the previous section, we can use the inference pair $< \{headache, runnynose\}, \{chest_congestion\} >$.

The non-sequitur part of an inference pair serves an important purpose. The inclusion of q as a non-sequitur is very different in semantics to $\neg q$ being part of the consequence. The later requires that there be an explanation for $\neg q$ whereas the former implies that a given explanation should not infer q. Consider that the doctor notices that a patient does not have a high body temperature and he or she includes non-high body temperature in the consequence set to be explained. This implies that the doctor needs an explicit explanation for the temperature being not high, which may be impossible to explain (since there are too many cases where the temperature need not be high). Including high body temperature as a non-sequitur avoids this need to explain non-high body temperature but making sure that high body temperature is not inferred from the diagnosis. The concept of non-sequitur is helpful in two ways: first, in avoiding explanation of the obvious as in the above example, and second, in constraining

explanations. For example, if one wants to find a path from point a to point b, but not through point c, then having $path(a, b)$ as part of consequence and $path(a, c)$ as part of non-sequitur constrains any plan (or explanation) generated to avoid going through c.

The consequence and the non-sequitur parts of an inference pair $< s, u >$ have semantically different meaning with respect to a theory T. Let S be an explanation for $< s, u >$. Whenever $p \in s$, then the explanation for p in S should be consistent with T. That is, if S infers p then T also should infer p. Moreover, if S uses a 'line of reasoning' to infer p, then the same line of reasoning should also be usable in T to explain p. On the other hand, whenever $q \in u$, S does not infer q, and it is possible that T infers q. That is, the lack of inference for q in S need not be consistent with T. In the example about path from point a to point b, it is possible that there are paths from a to c in the graph under consideration, but our plan (explanation) does not contain those paths.

Next, we make precise the definition of an explanation of an inference pair. (We do not include the subscript R in $\models$) We begin with the definition of a *weak explanation*. A weak explanation for an inference pair (s, u), can be seen as any subset of T which explains s and does not explain u, but any other positive inferences made from the explanation may not be consistent with theory T.

Definition 1 *A weak explanation e for an inference pair (s, u) from a theory T is defined as follows:*

1) $e \subseteq T$,

2) $e \models s$,

3) $\forall b \in u,\ e \not\models b$,

4) $\forall k((e \cup k \not\models s) \Rightarrow (T \not\models k))$ *where k is a set of atoms.*

The set of weak explanations for a pair (s, u) from a theory T is denoted as $E_w^T(s, u)$.

When $E_w^T(s, u)$ is empty it implies that the pair (s, u) is not explainable from T. Item (1) states that explanations are going to be formed from the initial theory. That is no other external facts are part of an explanation. Items (2) and (3) states that the explanations should conform to what it explains. That is, the explanation should derive s and should not derive u. Item (4) states that e is a consistent explanation for s from the underlying theory T. Items (1) and (4) are vital and make sure that the explanation for s is consistent with T. Item (4) makes sure that whenever a negation of an atom needs to hold in e for the proof of s, then that negation also holds in T. Item (1) makes sure that axioms used in the derivation of s occur in T also. Items (1) and (4) together make sure that the derivation of s in e can be mimicked in T. The purpose of (4) can be seen from the following example and Lemma 1.

Example 1 *Consider a theory $T = \{a \leftarrow b, \neg c;\ a \leftarrow b, \neg d;\ \ k \leftarrow \neg c;\ \ b;\ \ c\}$ and the pair $(\{a\}, \emptyset)$. Consider that the reasoning model is given as in logic programming, i.e., SLDNF-resolution. Then if (4) is ignored in Definition 1 then we can see $\{a \leftarrow b, \neg c;\ b\}$ is a possible explanation for $(\{a\}, \emptyset)$, but it is inconsistent with T, since there is no SLDNF-resolution for a using $a \leftarrow b, \neg c$ and b.*

since c is provable in T. By including Item (4) of the derivation, this inconsistent explanation is avoided.

Note that the definition of weak explanation does not avoid inferring other information which may be inconsistent with T. In the above example, $\{a \leftarrow b, \neg d;\ k \leftarrow \neg c;\ b\}$ is a weak explanation for $(\{a\}, \emptyset)$. The atom k is derivable from the explanation using SLDNF-resolution, whereas k is not similarly derivable from T. The derivation of inconsistent results from an explanation can be avoided using the following definition of a strong explanation.

Definition 2 *A strong explanation e for an inference pair (s, u) from a theory T is defined as follows:*
Items 1) through 4) are as given in Definition 1
5) $\forall t(e \models t \Rightarrow T \models t)$ where t is any arbitrary set of positive inferences.
The set of strong explanations for a pair (s, u) from a theory T is denoted as $E_s^T(s, u)$.

The strengthening by Item (5) makes the definition of strong explanation biased towards positive inferences and against negation. Even though a strong explanation has an attractive property that all positive inferences made from it are consistent with T, the construction of all strong explanations is more complicated compared to construction of all weak explanations. But, in practical cases, we are interested in constructing only a subset of the explanations and particularly minimal explanations, and such explanations are strong. Note that a strong explanation is also a weak explanation but not vice versa. Next we study some properties of weak explanations which are also inherited by strong explanation. The proofs of these properties can be found in [6].

Lemma 1 *If e is a weak explanation for an inference pair (s, u) from a theory T then $T \models s$.*

When a theory T is consistent then, if there is an explanation for a fact from T, then there is no explanation for its negation. (We define $\neg s = \{\neg C \mid C \in s\}$.)

Lemma 2 *Let T be a consistent theory and let (s, u) be a pair.*
Then, $E_w^T(s, u) \neq \emptyset \Rightarrow E_w^T(\neg s, \emptyset) = \emptyset$.

Any explanation for a given inference pair should not be affected by reducing theory T to a limited extent. Also whenever an inference pair is expanded to a limited extent, it should not affect the explanation. These robustness properties of an explanation are captured by the following lemma:

Lemma 3 *: Let T be a theory. Let (s, u) be a pair and $e \in E_w^T(s, u)$. Then the following hold:*
1) If $e \subseteq T' \subseteq T$ then $e \in E_w^{T'}(s, u)$.
2) If $e \not\models a$ then $e \in E_w^T(s, u \cup \{a\})$.

Note that $e \models t$ does not imply $e \in E_w^T(s \cup t, u)$. But for strong explanations we trivially have:

Lemma 4 *: Let T be a theory. Let (s,u) be a pair and $e \in E_s^T(s,u)$. If $e \models t$ then $e \in E_s^T(s \cup t, u)$.*

Next we derive some useful results concerning the non-existence of an explanation. In the case of explanations which do not have the non-sequitur counterpart (as in traditional explanations), if T proves the set of facts that need to be explained then there is an explanation for the set of facts. This does not hold in the case of controlled explanation, since if every explanation from T is constrained by some atom in the non-sequitur then there can be no controlled explanation possible.

Lemma 5 *: Let T be a theory and let (s,u) be a pair. Then*
$E_w^T(s,u) = \emptyset \Leftrightarrow (\forall T' \subseteq T,\ (T' \models s \rightarrow \exists b \in u(T' \models \{b\})))$

The following shows when there is no effect of the the constraining action of u.

Corollary 1 *If $T \not\models b,\ \forall b \in u$ then*
1) $(T \models s \Rightarrow E_w^T(s,u) \neq \emptyset)$
2) $(T \models s \Rightarrow T \in E_w^T(s,u)$ *and* $T \in E_s^T(s,u))$

3 Selection of Explanations

Even though the use of constraints restricts the number of explanations that are generated, one can still arrive at more than one explanation for a given phenomenon. In such cases, we adapt the techniques which are traditionally used for ordering the explanations and preferring some over others. In this section, we define two such types of explanations, set-minimal controlled explanations and cardinality-minimal controlled explanations. In [6] we develop other types of selection criteria.

One of the main issues in explanation generation is that of generating minimal explanations; that is explanations which use the smallest possible extent of the theory to explain an inference pair. The rationale behind the criteria of minimality is that one prefers the simplest explanation possible; the principle of "Occam's Razor". There are several ways of defining minimality. We start with one such definition:

Definition 3 *Let T be a theory and let (s,u) be an inference pair. Let $t \in E_w^T(s,u)$. Then t is called a* set-minimal explanation *for (s,u) using T if there are no other explanation $t' \in E_w^T(s,u)$ such that $t' \subset t$.*

For a given inference pair, there can be more than one set-minimal explanation and we refer to the set of set-minimal explanations for (s,u) using T as $E_{smin}^T(s,u)$. It can be seen that $E_{smin}^T(s,u) \subseteq E_w^T(s,u)$

Example 2 *Consider the theory T given as*
(1) pavement_wet $\leftarrow$ rained
(2) grass_wet $\leftarrow$ rained
(3) grass_wet $\leftarrow$ sprinkler_on

(4) rained
(5) sprinkler_on
We want to explain the inference pair $(s, u) = (\{grass_wet\}, \emptyset)$ *using* T. *The following explanations are set-minimal.*

$e_1 = \{(2), (4)\}$
$e_2 = \{(3), (5)\}$

Whereas, the following explanation is not set-minimal:

$e_3 = \{(1), (2), (4)\}$

In fact, there are 14 explanations for (s, u) *using* T, *but only* e_1 *and* e_2 *are set-minimal.*

Necessary and sufficient condition for the non-existence of a set-minimal controlled explanation is provided by the following lemma.

Lemma 6 *Let* T *be theory and let* (s, u) *be an inference pair.*
Then $E^T_{smin}(s, u) = \emptyset$ *if and only if* $E^T_w(s, u) = \emptyset$.

The following theorem shows that a set-minimal explanation is also a strong explanation.

Theorem 1 *Let* T *be a theory and let* (s, u) *be an inference pair.*
If $e \in E^T_{smin}(s, u)$ *then* $e \in E^T_s(s, u)$.

Next we provide the definition for cardinality-minimal controlled explanation.

Definition 4 *Let* T *be theory, let* (s, u) *be an inference pair and let* $t \in E^T_w(s, u)$. *Then* t *is called a* size-minimal explanation *(or* cardinality-minimal explanation*) for* (s, u) *using* T *if there are no other explanation* $t' \in E^T_w(s, u)$ *such that* $|t'| < |t|$.

For a given inference pair, there can be more than one size-minimal explanation and we refer to the set of size-minimal (or cardinality-minimal) explanations for (s, u) using T as $E^T_{cmin}(s, u)$. In Example 2, the explanations e_1 and e_2 are also size-minimal explanations for (s, u) using T, since all other explanations have a higher cardinality. The following lemma shows the relationship between set-minimal and size-minimal explanations.

Lemma 7 *Let* T *be a theory and let* (s, u) *be an inference pair.*
Then, $E^T_{cmin}(s, u) \subseteq E^T_{smin}(s, u)$.

4 Minimal Explanation Systems

In the previous sections we saw the definition for weak, strong, set-minimal and cardinality minimal controlled explanations. In this section, we provide a procedure for generating set-minimal controlled explanations. In [6] we provide a procedure for generating weak explanations and cardinality-minimal explanations.

We restrict our attention in this section to propositional Horn theories. This restriction can be lifted without affecting the soundness and completeness of the procedure. But the simplification helps in keeping our algorithms and proofs simple without any substitutions or unifiers being involved in them. We also consider that T is body-minimal which is defined as follows:

Definition 5 *Let T be a propositional Horn theory. Then, T is body-minimal, if there are no two rules $R_1 = A \leftarrow B_1, \ldots, B_n$ and $R_2 = A' \leftarrow C_1, \ldots, C_m$ in T such that $A = A'$ and $\{B_1, \ldots, B_n\} \subseteq \{C_1, \ldots, C_m\}$.*

If a theory is not body-minimal, it can easily be converted into one, by deleting appropriate rules. In the rest of the paper, we consider only theories that are body-minimal propositional Horn theories. We begin with a notion of an exp-goal.

Definition 6 *An* exp-goal *is a function of the form $exp(T, s, u, t)$ where T is a body-minimal propositional Horn theory, $t \subseteq T$, s is a set of literals, u is a set of atoms and $s \cap u = \emptyset$. When $s = \emptyset$ then the exp-goal is called an* empty exp-goal.

An esmin-derivation constructs a minimal explanation for a given inference pair. It operates on exp-goals.

Definition 7 *Let T be a body-minimal propositional Horn theory, (s, u) an inference pair. Then an* esmin-derivation *for (s, u) from T is a sequence of exp-goals $G_0 = exp(T, s, u, \emptyset), G_1, \ldots$ where each G_{i+1} is derived from G_i (for $i > 0$), as follows: Let $G_i = exp(T, s \cup \{L\}, u, t)$ be an exp-goal where L is a literal, called the selected literal. Then the exp-goal G_{i+1} is e-derived from G_i as follows:*

Case 1: $L = A$ is an atom and $A \leftarrow B_1, \ldots, B_n$ is a rule in T such that
$\forall i, 1 \leq i \leq n, (B_i \notin u \wedge B_i \neq A \wedge \nexists (B_i \leftarrow C_1 \ldots, C_r \in t))$:
$G_{i+1} = exp(T, s \cup \{B_1, \ldots, B_n\}, u, t \cup \{A \leftarrow B_1, \ldots, B_n\})$

Case 2: $L = \neg B$ and every esmin-derivation from $exp(T, \{B\}, \emptyset, \emptyset)$ fails*:*
$G_{i+1} = exp(T, s, u, t)$

Definition 8 *Let T be a body-minimal propositional Horn theory and let (s, u) be an inference pair. Then a* failed esmin-derivation *from $G_0 = exp(T, s, u, \emptyset)$ is an esmin-derivation which ends in an exp-goal $G_j = exp(T, s', u', t)$ where s' is non-empty and where neither of the two cases is applicable to the selected literal.*

Definition 9 *Let T be a body-minimal propositional Horn theory and let (s, u) an inference pair. Then an* esmin-derived explanation *t for (s, u) from T is given by a finite e-derivation of an empty exp-goal $exp(T, \emptyset, u, t)$ from $G_0 = exp(T, s, u, \emptyset)$. If $G_n = exp(T, \emptyset, u, t)$ we say that the derivation has length n.*

The explanation generated by Definition 7 is sound and complete with respect to the set-minimal explanation defined in Definition 3. The soundness is shown by first showing that a esmin-refutation provides a weak-explanation and then Then by showing that the explanation derived is minimal. The proofs for the following theorems are given in [6].

Theorem 2 *(Soundness Theorem) Let T be a body-minimal propositional Horn theory and let (s, u) be an inference pair. Let t be an esmin-derived explanation Then $t \in E_w^T(s, u)$.*

Theorem 3 *(Minimality Theorem) Let T be a body-minimal propositional Horn theory and let (s, u) be an inference pair. Let t be an esmin-derived explanation. Then $t \in E_{smin}^T(s, u)$.*

Theorem 4 *Let T be a body-minimal propositional Horn theory. and let (s, u) be an inference pair. Let $t \in E_{smin}^T(s, u)$. Then there is an esmin-derivation $G_0 = exp(T, s, u, \emptyset), \ldots, G_n = exp(T, \emptyset, u, t\})$ for some finite n.*

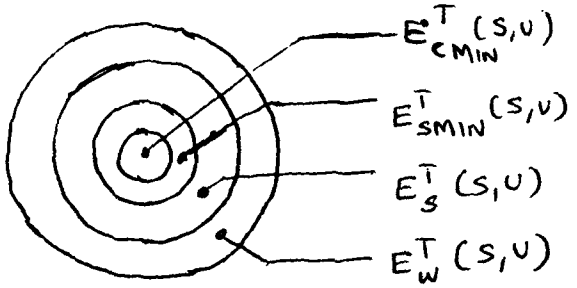
5 Discussion

The controlled explanation defined in this paper provides an additional method for restricting explanations generated by a system. One is interested in finding what is the effect of the constraint on the explanations; i.e., the effect of u in the inference pair $< s, u >$ on the generation of strong, weak and nunimal explanations. The following lemma holds.

Lemma 8 *Let T be a theory and Let (s, u) be a pair. Then*
1) $E_w^T(s, u) \subseteq E_w^T(s, \emptyset)$.
2) $E_s^T(s, u) \subseteq E_s^T(s, \emptyset)$.

The property does not hold for either set-minimal or size-minimal explanations. The reason being that one can define a set u such that every minimal explanation in $E_{smin}^T(s, \emptyset)$ is not in $E_{smin}^T(s, u)$. Hence, the set of set-minimal controlled explanation need not be a subset of uncontrolled set-minimal explanations. The same also holds for size-minimal explanations.

The following figure shows the relationship between various types of explanations defined in this paper.



Acknowledgement: We wish to express our appreciation to the National Science Foundation for their support of our work under grant number CCR-9110721.

Appendix

Consider the theory T given as

(1) $pavement_wet \leftarrow rained$
(2) $pavement_wet \leftarrow overflow$
(3) $grass_wet \leftarrow rained$
(4) $grass_wet \leftarrow sprinkler_on, tap_on$
(5) $rained \leftarrow cloudy_earlier$
(6) $tap_on \leftarrow \neg tap_off$
(7) $overflow \leftarrow sprinkler_on$
(8) $cloudy_earlier$
(9) $sprinkler_on$

The above theory provides rules for pavement and grass being wet. No information about the tap being off is provided in the theory and hence can be assumed to be false by the closed world assumption. Now consider that you want an explanation for the grass and the pavement being wet. Also assume that you have noticed that it was not cloudy earlier in the day. Hence you want to circumscribe your explanations not to include this information as part of the explanation. This can be done using the inference pair $(s, u) =< \{grass_wet, pavement_wet\}, \{cloudy_earlier\} >$ which needs to be explained using T. The following is an esmin-derivation providing a set-minimal explanation.

$G_0 = exp(T, \{\underline{grass_wet}, pavement_wet\}, \{cloudy_earlier\}, \emptyset)$.
$G_1 = exp(T, \{\underline{pavement_wet}, sprinkler_on, tap_on\}, \{cloudy_earlier\}, \{(4)\})$
$G_2 = exp(T, \{\underline{overflow}, sprinkler_on, tap_on\}, \{cloudy_earlier\}, \{(4), (2)\})$
$G_3 = exp(T, \{\underline{sprinkler_on}, tap_on\}, \{cloudy_earlier\}, \{(4), (2), (7)\})$
$G_4 = exp(T, \{sprinkler_on, \underline{\neg tap_off}\}, \{cloudy_earlier\}, \{(4), (2), (7), (6)\})$
$G_0' = exp(T, \{\underline{tap_off}\}, \emptyset, \emptyset)$
failure since all cases in Definition 7 are inapplicable and $s_0 \neq \emptyset$.
$G_5 = exp(T, \{\underline{sprinkler_on}\}, \{cloudy_earlier\}, \{(4), (2), (7), (6)\})$
$G_6 = exp(T, \emptyset, \{cloudy_earlier\}, \{(4), (2), (7), (6), (9)\})$

Hence $\{(4), (2), (7), (6), (9)\}$ is a minimal explanation for the query.

References

1. J. de Kleer. An Assumption-based TMS. *Artificial Intelligence*, 28:127-162, 1986.
2. J. de Kleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97-130, 1987.
3. J. Doyle. Truth Maintenance System. *Artificial Intelligence*, 13, 1980.
4. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.
5. D. Poole. A Logic for Default Reasoning. *Artificial Intelligence*, 36:27-47, 1988.
6. A. Rajasekar. Semantics of Explanation Systems, 1992. Submitted.
7. J.A. Reggia, D.S. Nau, and Y. Wang. Diagnostic Expert Systems based on a Set-Covering Model. *Int. J. on Man-Machine Studies*, 19:437-460, 1983.
8. R. Reiter. A theory of diagnosis from first principles. Technical report, Computer Science Department University of Toronto, 1986.

Signed Formulas: A Liftable Meta-Logic for Multiple-Valued Logics †

Neil V. Murray and *Erik Rosenthal*

Dept. of Computer Science
State Univ. of N.Y. at Albany
Albany, NY 12222
nvm@cs.albany.edu

Dept. of Mathematics
University of New Haven
West Haven, CT 06516
brodsky%nhu.UUCP@yale.edu

ABSTRACT. We consider means for adapting classical deduction techniques to multiple-valued logics. Some recent work in this area, including our own, utilizes signs (subsets of the set of truth values). In this paper we develop a language of signed formulas that may be interpreted as a meta-level logic. Questions not expressible in the underlying logic are easily expressed in the meta logic, and they may be answered with classical techniques because the logic is classical in nature.

We illustrate the applicability of classical techniques by specifically developing resolution for signed formulas. The meta logic admits a version of Herbrand's Theorem; as a result, these results extend naturally to the first order case. The fact that path resolution, path dissolution, and analytic tableaux are easily adapted to signed formulas, and that annotated logics are a special case of signed formulas is briefly discussed.

1. Introduction

In recent years a number of researchers have studied a variety of non-standard logics ([1,2,3,4,5,7,8,9,10,14,15,16,17,18,22], for example), largely as tools for investigating areas such as modeling uncertainty or natural language processing. These applications require computational deductive techniques. One that several authors have adapted is resolution [2,4,9,10,15,16,22]; others [1,3,5,7,8,14,18] have found the tableau method suitable, largely because it can produce lists of models. Our path dissolution rule [12,13], which generalizes the tableau method, also produces lists of models, and we have been able to adapt it to a class of multiple-valued logics (MVL's). In this paper we describe methods for applying these techniques to a much wider class of logics.

One feature common to much of this work, including our own, is the use of signs (subsets of the set of truth values). An adaptation of standard tableau signs for MVL's was first proposed by Surma [21] at the Int. Symp. on Multi-Valued Logics in 1974 and by Suchoń [20] that same year. In 1987 Carnielli extended Surma's work to a wider class of logics [1]. In 1990 Doherty [3] developed a tableau system for a variant of a three-valued logic developed by Sandewall [17]. The notion of a *set of truth values* as a sign is present in both Suchoń's and Doherty's work, but only implicitly. The explicit and formal development of sets-as-signs was introduced independently by Hähnle [7,8] and by the authors [14]. (Gabbay's labeled deductive systems [6] are quite related philosophically, but not technically.)

† This research was supported in part by National Science Foundation grants CCR-9101208 and CCR-9202013.

This approach allows the utilization of classical techniques for the analysis of non-standard logics. This is appealing since, at the meta-level, human reasoning is essentially classical. That is, regardless of the set of truth values associated with the logic, at the meta-level humans interpret statements about the logic to be either true or false. Consider, for example, a logic with three truth values, $\{0,\frac{1}{2},1\}$. We may wish to determine whether a formula $\mathcal{F}$ is satisfiable; i.e., whether it can evaluate to 1. Alternatively, one may be interested in whether $\mathcal{F}$ can evaluate to other than false, i.e., to $\frac{1}{2}$ or 1. These queries are captured by the signed formulas $\{1\}:\mathcal{F}$ and $\{\frac{1}{2}, 1\}:\mathcal{F}$. The answer to such queries is yes or no; that is, either the formula can so evaluate or not. Observe that both the queries and the answers are at the meta-level; observe also that these questions cannot even be formulated at the object-level.

When the underlying logic is classical, the distinction between meta- and object-levels becomes blurred because both employ what would appear to be the same set of truth values. Of course, true at the meta-level is a positive answer to a query about theoremhood or satisfiability, whereas, at the object-level, true is just one element of the boolean domain.

In this paper, we formalize the notion of signed formulas and their relationship to the base logic from which they are built. Given an MVL Λ, we introduce a new logic Λ_s whose atoms are signed formulas. The advantage of working with Λ_s is twofold: First, a variety of meta-level questions about those truth values to which formulas in Λ can evaluate are directly expressible in Λ_s. Secondly, many standard inference techniques can easily be adapted to apply to signed formulas.

In the next section the broad class of MVL's to be considered is formally defined, and a formal basis is provided for Λ_s, the language of signed formulas. We also show how Λ_s extends naturally to the first order case. In Section 3, we show how a very general version of the standard resolution rule applies to signed formulas. We conclude by summerizing how the annotated logics of da Costa, Henschen, Lu, and Subrahmanian [2] and of Kifer and Subrahmanian [9], may be regarded as special cases of signed formulas. We also indicate how other techniques such as the tableau method [19] and path dissolution [12,13] can be adapted to Λ_s.

2. Multiple-Valued Logics and Signed Formulas

We assume a (propositional) language Λ consisting of logical formulas built in the usual way from a set $\mathcal{A}$ of atoms, a set Γ of connectives, and a set χ of logical constants. For the sake of completeness, we precisely define a *formula* in Λ as follows:

1. Atoms are formulas.
2. If Θ is a connective of arity n and if $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n$ are formulas, then so is $\Theta(\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n)$.

Associated with Λ is a set Δ of truth values, and an interpretation for Λ is a function from $\mathcal{A}$ to Δ; i.e., an assignment of truth values to every atom in Λ. A connective Θ of arity n denotes a function $\Theta: \Delta^n \rightarrow \Delta$. Interpretations are extended in the usual way to mappings from formulas to Δ. Alternatively, a formula $\mathcal{F}$ of Λ can be thought of as denoting a mapping from interpretations to Δ.

Many authors are concerned with logics in which there is considerably more structure. For example, a common requirement is that the domain of truth values Δ be a complete distributive lattice in which $0 = glb(\Delta)$ and $1 = lub(\Delta)$, and that the the meet and join operations play the roles of conjunction and disjunction, respectively. The results in this paper do not depend on any such restrictions.

We use the term *sign* for any (expression that denotes a) subset of Δ. We define a *signed formula* to be an expression of the form $S:\mathcal{F}$, where S is a sign and $\mathcal{F}$ is a formula in Λ. We are interested in signed formulas because they represent queries of the form, "Are there interpretations under which $\mathcal{F}$ evaluates to a truth

value in S?" In a refutational theorem proving setting for classical logic, the query is typically {true}:$\mathcal{F}$, where $\mathcal{F}$ is the negation of a goal or conclusion, conjoined with some axioms and hypotheses; the answer hoped for is no. To answer arbitrary queries, we map formulas in Λ to formulas in a classical propositional logic Λ_s.

We call Λ_s the *language of signed formulas* and define it as follows: The atoms are signed formulas and the connectives are (classical) conjunction and disjunction. We emphasize that a signed formula S:$\mathcal{F}$ is an atom in Λ_s regardless of the size or complexity of $\mathcal{F}$ and thus *has no component parts in the language* Λ_s. The set of truth values is of course {true, false}.

2.1. Λ-Consistent Interpretations

An arbitrary interpretation for Λ_s may make an assignment of true or false to any signed formula (i.e., to any atom) in the usual way. Our goal is to focus attention only on those interpretations that relate to the sign in a signed formula. To accomplish this we define a Λ-*consistent interpretation* I_s for Λ_s to be an interpretation for which there exists an interpretation I for Λ such that for each atom S:$\mathcal{F}$, S:$\mathcal{F}$ is true under I_s if and only if the formula $\mathcal{F}$ in Λ is mapped into S by I. Intuitively, Λ-consistent means an assignment of true to all signed formulas whose signs are simultaneously achievable via some interpretation over the original language. If $\mathcal{F}_1$ and $\mathcal{F}_2$ are formulas in Λ_s, we write $\mathcal{F}_1 \models_\Lambda \mathcal{F}_2$ if whenever I_s is a Λ-consistent interpretation and $I_s(\mathcal{F}_1)$ = true, then $I_s(\mathcal{F}_2)$ = true.

The following lemma is immediate since each interpretation in Λ maps a formula to exactly one element of Δ.

Lemma 1. Let I_s be a Λ-consistent interpretation, let A be an atom and $\mathcal{F}$ a formula in Λ, and let S_1 and S_2 be signs. Then:

i) $I_s(\varnothing : \mathcal{F})$ = false;

ii) $I_s(\Delta : \mathcal{F})$ = true;

iii) $S_1 \subseteq S_2$ if and only if $S_1 : \mathcal{F} \models_\Lambda S_2 : \mathcal{F}$ for all formulas $\mathcal{F}$;

iv) There is exactly one $\delta \in \Delta$ such that $I_s(\{\delta\} : A)$ = true. □

Although we feel that the focus on Λ-consistent interpretations is intuitive, it can also be motivated by the following technical observation: The formulas under consideration in Λ_s do not have any occurrences of negation; only $\wedge$ and $\vee$ appear as classical interpreted symbols. Whether or not such formulas are satisfiable with respect to Λ-consistent interpretations, they are trivially satisfiable with respect to arbitrary interpretations.

2.2. Λ-atomic formulas

Many classical inference rules begin with links (complementary pairs of literals). Such rules typically deal only with formulas in which all negations are at the atomic level. Similarly, the inference techniques that we wish to develop here require that signs be at the "atomic level." To that end, we call a formula Λ-*atomic* if it has the property that whenever S:A is an atom in the formula, then A is an atom in Λ; we call it *elementary* if whenever S:A is an atom, S is a singleton. Lemma 2 describes a method for driving signs inward for a restricted class of MVL's; repeated applications eventually produces a Λ-atomic formula. The lemma is immediate since the right side of the equation amounts to an enumeration of the interpretations that map the formula on the left side into S.

Lemma 2. Suppose that the truth domain Δ of an MVL Λ is finite. Let Θ be a connective in Λ of arity n, let $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n$ be formulas in Λ, and let S be a sign. Then if I_s is any Λ-consistent interpretation,

$$I_s\left(S : \Theta(\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n)\right) = I_s\left(\bigvee_{\langle \delta_1, \cdots, \delta_n \rangle \in \Theta^{-1}(S)} \left(\bigwedge_{i=1}^{n} (\{\delta_i\} : \mathcal{F}_i)\right)\right). \quad \square$$

Observe that the end product of repeated applications of the lemma will be elementary; this may be undesirable because this representation may be unnecessarily large. The Reduction Lemma in Section 3 may yield a more efficient representation, and knowledge of the connective Θ (for a particular logic) may help to drive the sign inward in a more efficient manner. When Δ is infinite, unless some equivalent of Lemma 2 is available, attention must be restricted to Λ-atomic formulas. These questions are discussed in greater detail in Section 3.

Remark: There is a natural one-to-one correspondence between interpretations over Λ and Λ-consistent interpretations over Λ_s as follows: Given an interpretation I over Λ, define the interpretation I_s over Λ_s by $I_s(\delta: A) = \text{true}$ iff $\delta = I(A)$. Since connectives in Λ are functions, I_s uniquely extends to an interpretation over all atoms of Λ_s. Conversely, by part iv of Lemma 1, the value of a Λ-consistent interpretation on elementary atoms determines a unique interpretation over Λ.

2.3. First Order Considerations

It is straightforward to introduce variable, constant, and function symbols into Λ with their usual meaning. We assume a non-empty domain of discourse $\mathcal{D}$. There appears to be no natural way to introduce quantifiers into Λ while preserving full generality. Consider, for example, the expression $\forall x\ P(x)$ in predicate calculus. It may be viewed as implicitly containing a sign, i.e., the quantified expression is true only if P maps all domain elements into {true}. The implicit sign in essence designates a subset of truth values that must contain the range of P(x) in order that the quantified expression denote true. We can generalize this notion by introducing the quantifiers $\forall^S$ and $\exists^S$: $\forall^S x\ P(x)$ denotes true only if P maps all domain elements into S, and $\exists^S x\ P(x)$ denotes true only if P maps some domain element into S. But these quantifiers denote true or false, not arbitrary elements of Δ, and hence fall more naturally within Λ_s.

We may instead view the predicate calculus expression $\forall x\ P(x)$ as denoting the greatest lower bound of the range of P(x) (where the boolean domain is the obvious two element lattice). This does not require the concept of sign, and does generalize to Λ. Thus we may define quantifiers $\forall$ and $\exists$ as the *infimum* and *supremum*, respectively, of the denotations of their arguments; of course, this is possible only if Δ is a lattice. We leave a more thorough exploration of these issues to a later paper.

If we do restrict quantifiers to Λ_s, then all variable occurrences in Λ are free. Formulas having n free variables then denote functions from $\mathcal{D}^n$ to Δ.

It is easy to see that for an arbitrary interpretation over any domain of discourse $\mathcal{D}$ and any truth domain Δ, a corresponding Herbrand interpretation can be constructed over the Herbrand Universe defined in the usual way. Ground atoms are simply partitioned according to their denotations in Δ. As a result, the notion of a Λ-consistent interpretation is still meaningful if we extend Λ_s to include the quantifiers $\forall$ and $\exists$. In classical logic, Herbrand's Theorem can be proved by applying König's Lemma to semantic trees, and it goes through in the same way in Λ_s with respect to Λ-consistent interpretations.

Since extending Λ_s to be a classical first order logic raises no truly new issues, we restrict discussion to the ground case for the remainder of the paper.

3. Signed Inference

In this section, we adapt resolution to produce an inference rule for Λ_s. The techniques developed in [14] and in [15] for dealing with signed formulas from a fairly specialized class of logics also apply to Λ-atomic formulas. The basic idea is the next lemma, which follows immediately from part iv of Lemma 1. First, we say that two formulas $\mathcal{F}_s$ and $\mathcal{F}'_s$ in Λ_s are Λ-*equivalent* if $I_s(\mathcal{F}_s) = I_s(\mathcal{F}'_s)$ for any Λ-consistent interpretation I_s; we write $\mathcal{F}_s \equiv_\Lambda \mathcal{F}'_s$.

Lemma 3 (*The Reduction Lemma*). Let $S_1{:}A$ and $S_2{:}A$ be Λ-atomic atoms in Λ_s; then $S_1{:}A \wedge S_2{:}A \equiv_\Lambda (S_1 \cap S_2){:}A$ and $S_1{:}A \vee S_2{:}A \equiv_\Lambda (S_1 \cup S_2){:}A$. □

3.1. Notation

The formulas in Λ_s that we are interested in are in *negation normal form* (NNF): The only connectives used are conjunction and disjunction. (NNF also requires that negations must reside at the atomic level. This is irrelevant here since negation is absent altogether from the formulas we are considering.) Conjunctive and disjunctive normal forms (CNF and DNF) are special cases of NNF; considerable time and space may be required to put NNF formulas into CNF or DNF.

We have found that NNF formulas naturally lend themselves to a two-dimensional representation. For example, in Figure 1 below, the formula on the left is displayed graphically on the right:

$$((\neg C \wedge A) \vee D) \wedge (\neg A \vee (B \wedge C)) \quad = \quad \begin{array}{ccc} \overline{C} & & \\ \wedge & \vee & D \\ A & & \end{array} \quad \wedge \quad \begin{array}{ccc} & & B \\ \overline{A} & \vee & \wedge \\ & & C \end{array}$$

Figure 1.

Disjunctions are displayed horizontally, conjunctions vertically. A formula so represented is called a *semantic graph*; for a detailed exposition, see [11].

Since conjunction (and disjunction) are commutative and associative, when viewed as a graph, their arguments are called *fundamental subgraphs*. The fundamental subgraphs of the upper disjunction in the graph above are $\begin{array}{c} C \\ \wedge \\ A \end{array}$ and the literal D.

The graph above contains four *c-paths* (maximal conjunctions of literal occurrences): {C, A, A}, {C, A, B, C}, {D, A}, {D, B, C}. We say that two literal occurrences are *c-connected* if they are both in some c-path. With respect to arbitrary interpretations, a semantic graph is unsatisfiable if and only if every c-path is unsatisfiable, and a c-path is unsatisfiable if and only if it contains a link. The formulas we study in Λ_s have no links in the classical sense, but, as we shall see, the notion of link can be redefined with respect to signs in a meaningful way.

3.2. Signed resolution

In this section we generalize the resolution inference rule for classical logic. Combined with the Reduction Lemma, this generalization will lead naturally to an inference rule for Λ_s.

Let $\mathcal{F}$ be a conjunction of disjunctions in a classical logic. We use the term clause for the disjunctions even though we do not assume them to be disjunctions of literals. Let two of the clauses be $C = Q_C \vee \left[\bigvee_{j=1}^{m_C} J_j\right]$ and $D = Q_D \vee \left[\bigvee_{j=1}^{m_D} K_j\right]$. Then we may *resolve* clauses C and D and infer

$$\begin{array}{c} Q_C \\ \wedge \\ Q_D \end{array} \vee \left[\bigvee_{j=1}^{m_C} J_j\right] \vee \left[\bigvee_{j=1}^{m_D} K_j\right].$$

Soundness follows from noting that if $C \wedge D$ is true, but none of the J_j's and none of the K_j's are, then Q_C and Q_D must both be true. Observe that when C and D are clauses of literals and Q_C are Q_D are complimentary, this rule reduces to standard binary resolution.

Consider now a Λ-atomic formula $\mathcal{F}$ in Λ_s that is in conjunctive normal form (CNF). As such, the clauses of $\mathcal{F}$ are sets of literals. Let $C_j, 1 \le j \le r$, be clauses in $\mathcal{F}$ that contain, respectively, Λ-atomic atoms $\{S_j:A\}$. Thus we may write $C_j = K_j \cup \{S_j:A\}$. Then the resolvent R of the C_j's is defined to be the clause $(\bigcup_{j=1}^{r} K_j) \cup \{(\bigcap_{j=1}^{r} S_j):A\}$; this definition is the one given above, generalized to n clauses and specialized to Λ-consistent interpretations via the Reduction Lemma.

We must augment this definition with the following obvious simplification rules that also stem from the Reduction Lemma. First, if $\bigcap_{j=1}^{r} S_j$ is empty, then $(\bigcap_{j=1}^{r} S_j):A$ is false and may simply be deleted from R. Secondly, whenever $S_1:B$ and $S_2:B$ are in R, we replace them by $(S_1 \cup S_2):B$; if $(S_1 \cup S_2) = \Delta$, then R is a tautology and may be deleted from $\mathcal{F}$.

The classical notion of subsumption also generalizes to Λ_s: clause C *subsumes* clause D if, for every literal $S:A \in C$, there is a literal $S':A \in D$ such that $S \subseteq S'$.

3.3. An example

Consider the logic Λ based on the three-valued domain $\{0,u,1\}$, ordered so that $0 < 1$ and $u < 1$, but 0 and u are not comparable. There are two unary connectives, $\neg$ and $\sim$, and two binary connectives $\otimes$ and $\oplus$; their truth tables are shown below.

	$\neg$	$\sim$
0	1	1
u	1	1
1	0	u

$\otimes$	0	u	1
0	0	0	0
u	0	u	u
1	0	u	1

$\oplus$	0	u	1
0	0	u	1
u	u	1	1
1	1	1	1

Observe that although the two binary connectives are "and-like" and "or-like", Δ is not a lattice.

Consider the formula

$$\mathcal{F} = (\sim A \oplus B) \otimes \neg (A \oplus B)$$

We may ask if $\mathcal{F}$ is "satisfiable," i.e., if it can possibly evaluate to 1, by considering the formula $\{1\}:\mathcal{F}$ in Λ_s. If $\{1\}:\mathcal{F}$ does indeed have Λ-consistent models, we may be interested in knowing them; this situation favors the use of analytic tableaux or path dissolution in the analysis, since these methods can effectively determine satisfying interpretations. Alternatively, if we suspect that $\mathcal{F}$ *always* evaluates to 1, then a refutation procedure such as resolution could be employed on a Λ-equivalent of $\{0,u\}:\mathcal{F}$.

For our example, we use the signed formula $\{1\}:\mathcal{F}$.

$$\{1\} : \mathcal{F} = \{1\} : (\sim A \oplus B) \otimes \neg (A \oplus B)$$

Since $\otimes^{-1}(1) = \{<1,1>\}$, Lemma 2 gives us:

$$\{1\} : (\sim A \oplus B)$$
$$\wedge$$
$$\{1\} : \neg (A \oplus B)$$

Direct application of Lemma 2 on both conjuncts of the above formula leads to the (not yet Λ-atomic) formula of Figure 2 below:

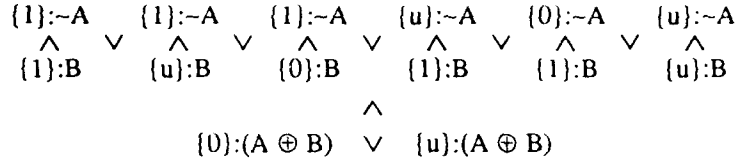


Figure 2.

The careful reader will have noticed that the formula above can be simplified. Factoring the leftmost three disjuncts in the upper half of the graph on the atom [{1}:~A] results in the formula [{1}:~A ∧ ({0}:B ∨ {u}:B ∨ {1}:B)]. This is just [{1}:~A ∧ Δ:B] by the Reduction Lemma, and, by part ii of Lemma 1, it reduces further to {1}:~A. Similarly, the three disjuncts containing {1}:B (using a copy of the leftmost disjunct) reduce to {1}:B.

From the discussion above, and from the application of Lemma 2 to the lower half of Figure 2, the formula in Figure 3 results.

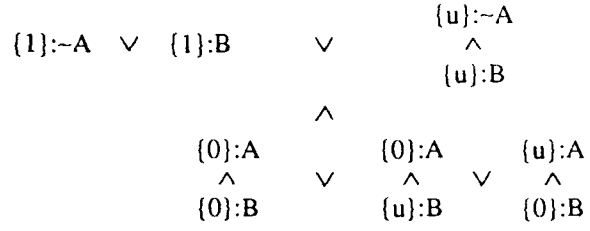


Figure 3.

By applying Lemmas 2 and 3 to the upper part of Figure 3, then factoring (on {0}:A, {u}:A, and finally on {0,u}:B) and applying Lemma 3 to the lower part, Figure 4 is produced.

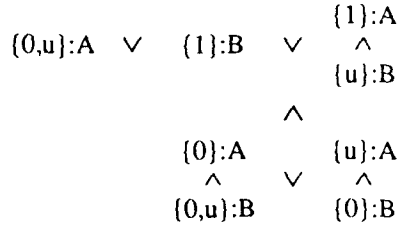


Figure 4.

The formula in Figure 4 is a conjunction but is not in CNF. We may convert the upper conjunct into the two clauses {{0,u}:A, {1}:B, {1}:A} and {{0,u}:A, {1}:B, {u}:B}, which are Λ-equivalent to true and {{0,u}:A, {u,1}:B}, respectively. Similarly, {{0,u}:B} is one of the clauses obtainable from the lower conjunct. As an example of a resolution step, resolving {{0,u}:A, {u,1}:B} and {{0,u}:B} produces {{0,u}:A, {u}:B}. Note that in this simple example, however,

the clause {{0,u}:A}, also obtainable from the lower conjunct, subsumes {{0,u}:A, {u,1}:B}; in effect, the upper conjunct itself is subsumed and could be dropped.

The example illustrates the explosive growth that can occur with applications of Lemma 2. Lemmas 1 and 3 can be used to reduce the size of the formula, but the process of applying Lemma 2 and then condensing the result is likely to be expensive. For a given connective in a given logic Λ, the simplifications available from the Lemmas may be pre-computable, producing rules that directly yield the more condensed formulas. Consider, for example, the truth table for $\oplus$. It is easy to see that the following rule is valid:

$$\{1\}:(A \oplus B) \equiv_{\Lambda} \{1\}:A \vee \{1\}:B \vee (\{u\}:A \wedge \{u\}:B) .$$

Even this rule can lead to exponential growth, but this is analogous to the cost of putting a classical formula into CNF.

More generally, the expense of driving signs inward depends on the efficiency of such rules. Lemma 1 is a very inefficient means of accomplishing this task; the above rule is more reasonable. With some connectives in some logics it may not be possible to develop efficient rules, but such cases are likely to reflect the inherent complexity of the base logic.

3.4. Completeness

The resolution rule presented above was introduced in [15] for a class of multiple-valued logics. There, the truth value domains were assumed to obey rather severe restrictions, and all queries had the form: Can a formula evaluate to 1 (the maximal element of a truth value lattice)? Refutation completeness was proven using a modified semantic tree argument in which interpretations over the MVL were represented.

In this paper, we are dealing with a much wider class of MVL's and are allowing more general queries about the formulas in those MVL's. The resolution rule and the signed formulas on which it operates, however, are defined in a classical logic. In this setting, completeness is not a statement about what formulas in Λ can be proved inconsistent; rather, completeness means that we can answer certain queries about formulas in Λ by deriving the empty clause in Λ_s. Nevertheless, the semantic tree construction in [15] can be adapted to produce a completeness argument for Λ_s.

Theorem 1. Let $\mathcal{F}$ be a formula in Λ, and let S be a subset of Δ such that no interpretation maps $\mathcal{F}$ into S. Then there is a derivation of the empty clause from any Λ-atomic equivalent of S:$\mathcal{F}$ using signed resolution. □

4. Conclusions

Signed resolution appears to provide a unifying framework for most adaptations of resolution to MVL's. For example, the annotated logics (also called *paraconsistent* logics) introduced by da Costa, Henschen, Lu, and Subrahmanian [2], and by Kifer and Subrahmanian [9] employ two inference rules, *p-resolution* and *reduction* (sometimes referred to as *mega-resolution* and *cloning*). Both rules are special cases of signed resolution. A question worthy of investigation is whether signed resolution yields some form of linear resolution for these logics.

The technique developed for applying resolution to MVL's via signed formulas works with most classical inference techniques. For example, *path resolution*, a generalization of resolution that permits resolving on sets of links in formulas in NNF, can easily be adapted to λ_s. Hähnle's work [7,8] with tableaux does this for the method of analytic tableaux, and path dissolution, of which the tableau method is

one special case, was similarly adapted in [15]. Space does not permit a detailed description in this paper, but the adaptations are quite straightforward because the logic of signed formulas is a classical logic.

Our approach in this paper is very general in that there are no requirements on the structure of the domain of truth values; logical connectives are similarly unrestricted. However, when Δ is infinite, we must begin with Λ-atomic formulas; for finite Δ, arbitrary formulas in Λ_s can in principle be handled, but this can lead to explosive growth when Lemma 2 is employed directly. It would be useful to investigate the relationship between conditions placed on Δ and on the connectives, and the resulting rules for deriving Λ-atomic formulas. Hähnle [8] has investigated this issue from the standpoint of analytic tableau procedures. It would also be useful to discover a class of logics for which Δ is infinite and for which some technique (analogous to Lemma 2) for driving signs inward can be developed.

Acknowledgements

Discussions with James Lu have been helpful in clarifying our understanding of annotated logics. We are grateful to him for pointing out the relationship between signed resolution and resolution for annotated logics. We have also benefited from discussions with Reiner Hähnle and with V.S. Subrahmanian.

References

1. Carnielli, W.A. Systematization of finite many-valued logics through the method of tableaux. *Journal of Symbolic Logic*, 52,2 (June 1987), 473-493.

2. da Costa, N.C.A., Henschen, L.J., Lu, J.J., and Subrahmanian, V.S. Automatic Theorem Proving in Paraconsistent Logics: theory and Implementation. *Proceedings of the 10-th International Conference on Automated Deduction*, Kaiserslautern, W. Germany, July 1990. In *Lecture Notes in Artificial Intelligence*, Springer Verlag, Vol. 449, 72-86.

3. Doherty, P. Preliminary report: NM3 – A three-valued non-monotonic formalism. *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems*, Knoxville, Tennessee October 25-27, 1990. In *Methodologies for Intelligent Systems*, **5** (Ras, Z., Zemankova, M., and Emrich, M. eds.) North-Holland, 1990, 498-505.

4. Fitting, M. Resolution for Intuitionistic Logic. In *Methodologies for Intelligent Systems*, (Ras, Z. and Zemankova, M., eds.) North-Holland, 1987, 400-407.

5. Fitting, M. First-order modal tableaux. *J. Automated Reasoning* **4**, (1988) 191-213.

6. Gabbay, D. *LDS – Labeled Deductive Systems*. Oxford University Press, to appear.

7. Hähnle, R. Towards an efficient tableau proof procedure for multiple-valued logics. *Proceedings of the Workshop on Computer Science Logic*, Heidelberg, 1990. In *Lecture Notes in Computer Science, Springer Verlag, Vol 533, 248-260.*

8. Hähnle, R. Uniform notation tableau rules for multiple-valued logics. *Proceedings of the International Symposium on Multiple-Valued Logic*, Victoria, BC, May 26-29, 1991, 238-245.

9. Kifer, M. and Subrahmanian, V.S. On the expressive power of annotated logic programs. *Proceedings of the 1989 North American conference on Logic Programming*, Cleveland, OH, 1069-1089.

10. Morgan, C. Resolution for many-valued logics. *Logique et Analyse* 74-76 (1976), 311-339.

11. Murray, N.V., and Rosenthal, E. Inference with Path Resolution and Semantic Graphs. *JACM* 34,2 (April 1987), 225-254.

12. Murray, N.V., and Rosenthal, E. Dissolution: Making paths vanish. To appear, *JACM*.

13. Murray, N.V., and Rosenthal, E. Employing path dissolution to shorten tableau proofs. *Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation*, Portland, Oregon July 17-19, 1989, 373-381.

14. Murray, N.V., and Rosenthal, E. Improving tableaux proofs in multiple-valued logic. *Proceedings of the* 21st *International Symposium on Multiple-Valued Logic*, Victoria, B.C., Canada, May 26-29, 1991, 230-237.

15. Murray, N.V., and Rosenthal, E. Resolution and path dissolution in multiple-valued logics. *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, October 16-19, 1991. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 542, 570-579.

16. O'hearn, P., and Stachniak, Z. Note on theorem proving strategies for resolution counterparts of non-classical logics. *Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation*, Portland, Oregon July 17-19, 1989, 364-372.

17. Sandewall, E. The semantics of non-monotonic entailment defined using partial interpretations. In M. Ginsburg, M. Reinfrank, and E. Sandewall, ed., *Non-Monotonic Reasoning, 2nd International Workshop*. Springer, 1988.

18. Schwind. A tableau based theorem prover for a decidable subset of default logic. *Proceedings* of the *10th International Conference on Automated Deduction*, Kaiserslautern, W. Germany, July 24-27, 1990. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 449, 528-542.

19. Smullyan, R.M. *First-Order Logic*. Springer Verlag, 1968.

20. Suchoń, W. La méthode de Smullyan de construire le calcul n-valent de Lukasiewicz avec implication et négation. *Reports on Mathematical Logic*, Universities of Cracow and Katowice, 1974, **2**, 37-42.

21. Surma, S.J. An algorithm for axiomatizing every finite logic. In *Computer Science and Multiple-Valued Logics*, David C. Rine, Ed., North-Holland, Amsterdam, 1984, 143-149.

22. Stachniak, Z. Note on resolution circuits. *Proceedings* of the *International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, October 16-19, 1991. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 542, 620-629.

New Design Concepts for the FLINS-Fuzzy Lingual System: Text-based and Fuzzy-centered Architectures

Shun'ichi Tano, Wataru Okamoto, and Toshiharu Iwatani
Laboratory for International Fuzzy Engineering Research
Siber Hegner Building 3F, 89-1 Yamashita-cho,
Naka-ku, Yokohama, 231, JAPAN

Abstract. A fuzzy natural language communication system called the Fuzzy Lingual System (FLINS) is currently under development at the Laboratory for International Fuzzy Engineering Research (LIFE). The final goal of the FLINS project is to create a lingual computer, that is, a domain-independent teach, question, and answer (TQA) system. In this paper, we propose two new design concepts, text-based architecture and fuzzy-centered architecture, to realize our goal. Two experimental systems were built to determine the feasibility of those approaches before the development of FLINS. One is a text-based natural language understanding system, called the AB-System. The other is a fuzzy expert system called FOREX, which predicts exchange rate trends according to fuzzy rules and fuzzy data.

1. Introduction

Even though computer systems have great computing power and can contain a great amount of information, current user interfaces are so rigid and unfriendly that only specialists can make full use of these systems. We at the Laboratory for International Fuzzy Engineering Research (LIFE) are developing a natural language communication system called FLINS, which is short for Fuzzy Lingual System to overcome this barrier. Our ultimate goal is to create a lingual computer that functions as a domain-independent teach, question, and answer (TQA) system. It is an old problem, but still alive.

FLINS is based on two new design concepts, text-based architecture and fuzzy-centered architecture. The former was directly derived from the problems associated with conventional natural language understanding systems, and can be viewed as an AI-related problem. The latter concept is a little different and may be quite new to people in the AI community.

2. Motivation and Basic Approach

2.1 Motivation

Our final goal is a lingual computer that facilitates the interaction between users and computer systems. The fundamental features are:

Communication by means of natural language

A user and system should be able to communicate by means of natural language.

Teaching by means of natural language

A user should be able to teach the system new knowledge by means of natural language. The new knowledge includes meta-knowledge for control of the system itself, rule-type knowledge, and simple factual information.

Learning by means of natural language

The system should be able to store all interaction between a user and the system in a knowledge base (which we call a text base because all interactions use text). The system should be able to manage unknown situations by using this text as case data.

2.2 Basic Approach

Our basic approach is to combine the three key technologies, knowledge engineering, natural language processing and fuzzy engineering. Here's a brief analysis of the successes and failures of the technologies.

(1) Knowledge Engineering

The key idea in knowledge engineering (KE) is "knowledge is power", and a new architecture for an intelligent system based on this idea was proposed. While the divided knowledge base and inference mechanism architecture has proven to be useful, the difficulty of knowledge representation prevents it from being used extensively.

(2) Natural Language Processing

SHRDLU is the most famous natural language processing system. It worked quite well in a block world. But the system was not generalized.

A number of machine translation systems are currently in use[7]. Although their translation capability is limited, they can transform most text into a case-based structure[3], and they accept an almost unlimited range of text. Moreover, these machine translation systems use very similar sets of "cases". This demonstrates that methodologies have been established almost up to the "case structure" level.

(3) Fuzzy Engineering

Fuzzy theory was proposed by L.A. Zadeh in 1965[12]. At first this theory was ignored but as the number of successful practical applications increased, especially in the area of a fuzzy control, this situation changed rapidly. It was originally proposed to deal with the fuzziness found in such natural language. So it provides us with a good framework for coping with fuzziness in natural language.

3. Problems with Conventional Systems [e.g., 1, 5, 9].

(1) Closed Nature

Actually, it is impossible to provide natural language understanding systems with every piece of knowledge, such as knowledge about the target field and meta-knowledge, as well as word definitions and sentence structure knowledge, as inherent knowledge. Therefore, it is very important that users can change what the system already knows and can teach the system what it does not know through natural language.

Such extensibility is a key attribute for a true natural language understanding system. However, most conventional systems do not allow users to use natural language to teach and change knowledge. Although a few systems that can accept knowledge in natural language exist, they can only accept simple knowledge and their treatment of the knowledge is ad hoc.

(2) Implicit and Inherent Knowledge Represented in a Special Form

Most systems have implicit and inherent knowledge represented in a special form, such as program source code (e.g., LISP code), frames and scripts. In some cases, special tricks are hidden in this manner.

The big problem with implicit knowledge is that the system looses transparency. Thus most natural language understanding systems work only for specific fields and are never applied in other fields. Furthermore the more intelligent the system, the more likely important knowledge is represented in a specific form as implicit knowledge (tricks) . The problem with representing knowledge in special form is that it is almost impossible for users who are not familiar with the special form to make changes in the knowledge or add new knowledge.

Note that we do not deny the effectiveness of representations such as rules and frames. We only insist that a user should be able to access the important knowledge, so it has

to be accessible by means of natural language instead of the special form. It is no problem that it is represented in a special form as long as it can be accessed by means of natural language.

(3) Using Deep Structure to Represent Meanings

Most natural language understanding systems adopt knowledge representation that is based on deep structures, such as CD [8] or logical forms[1]. In this representation, the meaning of the text is represented using deep semantic primitives. For example, "to give" is transformed into "to transfer the ownership to others". Of course, this representation is sound and is suitable for manipulation of the meaning, because it is a sort of canonical form of the meaning. However, there is one inherent problem with this type of representation. To transform text into a deep structure, the system must completely understand the meaning of the text, which is very difficult even for humans. Moreover, defining such complete and domain-independent sets of deep semantic primitives and the rules for transformation from text to deep structures is actually impossible.

We think this is the reason most existing natural language understanding systems can never be applied outside of a fixed field or the limited situations they were originally designed to handle. From the theoretical point of view, the concept of deep structure is attractive, but it is not powerful enough to devise a natural language understanding system that is applicable to various real fields.

(4) Ignorance of fuzziness

One essential characteristic of natural language is fuzziness. This fuzziness ranges from the ambiguity caused by multiple parse trees for one sentence or multiple meanings for one word to the fuzziness of the meaning itself. Although the former type of fuzziness has been thoroughly studied for disambiguation algorithms, the latter type has not. It has been ignored in studies of natural language processing so far.

4. New Design Concepts

4.1 Text-Based Architecture

The basic concept of the text-based architecture is that all knowledge should be presented as text. "All knowledge" includes linguistic knowledge (i.e., word definitions and knowledge of sentence structure), knowledge of the target field (i.e., domain-dependent rules and data), and meta-knowledge (i.e., how to use the knowledge itself and how to control the system with it). We refer to this collection of knowledge as a text base rather than as a knowledge base to express clearly that all knowledge is represented as text.

The system uses only a basic pattern matcher that is controlled by a schema described by text to process all knowledge. This feature allows the user to use text to teach even meta-knowledge because the process of problem solving is controlled by text.

The following four subconcepts make a text-based architecture feasible. They are indispensable to implementation of the text-based architecture. In other words, the text-based architecture mentioned above is a principal and the following four subconcepts are methodologies for implementing the principal.

(1) Combination of Case-based Structure and the Text Base

Current machine translation systems have limited translation capability, but they can transform most text into a case-based structure with almost no constraints on the input text. Moreover, these machine translation systems use very similar sets of cases, so it is quite natural to use a case-based structure for general knowledge representation.

The problem in doing so, however, is that the case-based structure is too sensitive to vocabulary and sentence structure, because words in the input text appear in the case-

based structure directly and the sentence structure sometimes affects the result. Therefore even if the meaning of two texts is the same, their case-based representations may differ.

However, if the system can accept linguistic knowledge, the problem of case-based representation can be solved. This is because if the system accepts linguistic knowledge as well as domain-dependent knowledge, it can infer the meaning of a text by using the linguistic knowledge in exactly the same way that domain-dependent knowledge is used for problem solving. The more linguistic knowledge a user teaches the system, the more deeply the system can understand a text.

The combination of case-based structure and linguistic knowledge is a nice alternative to deep structure. In the case of the deep semantic structure, text is transformed into a certain structure at the fixed depth of meaning and is used in problem solving as indicated by the dotted line shown in Fig. 1. For combined case-based structure and text base, the depth of meaning is not fixed at all, rather it changes according to the progress of problem solving.

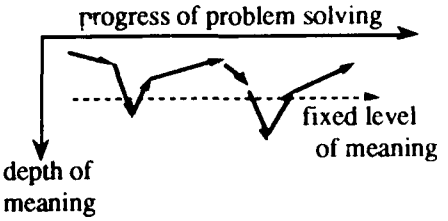


Fig. 1 Image of Text-based Inference

(2) Semantic and Procedural Primitives

All a text-based system knows initially are semantic and procedural primitives. Note that our semantic primitives are completely different from those of CD theory[8].

Semantic primitives are words that have a special role in inference. For example, "imply", "if" and "when" are semantic primitives. They are evaluated through inference when paraphrasing a text or making a new goal event.

Procedural primitives are connected subroutines and execute primitive functions, such as inference pattern matching and database calls. Words like "system-call" and "event-match" are examples of procedural primitives. If these procedural primitives are evaluated under certain conditions, the procedure is called automatically. Inference can be controlled by activating a procedural primitive. This means that a user can use procedural primitives to specify how to infer or how to answer. This feature enables the user to teach even meta-knowledge by using text. Moreover, it enables the system to explain its own behavior because the status of processing is also represented by case-based structure that can easily be transformed into natural language.

The inherent knowledge of the system represented in special form consists of only these two types of primitives. All other knowledge acquired through interaction with users is represented as text.

(3) Network Inference

All knowledge in a text-based system is represented in a case-based structure, which looks like an extended semantic network. The basic function of inference in network inference is modeled as an interaction among the networks. Inference patterns include direct matching, paraphrase matching, and deductive matching.

As mentioned above, all knowledge is represented as text. If texts in the text base are not combined into one network but rather exist as separate networks, the inference mechanism has to combine the text every time an input occurs, which could be a serious problem. The text in a text base should thus be combined into one network. This operation is equivalent to the precomputation of inference or a partial computation.

(1) Non-hierarchical Wondering Inference

The text base can be seen as shown in Fig. 2. The primitive layer includes words which have special meaning either semantically or procedurally. In the word layer, linguistic knowledge is stored. For example, Fig. 2 shows that the word "beheaded" is a form of the word "executed". It is possible to specify the constraints on the relation. For example, " to execute a person implies to kill a person" specifies that the object of each verb should be a person. This layer can be seen as a word dictionary. Domain-specific knowledge is stored in the knowledge layer and meta-knowledge is stored in the meta-knowledge layer.

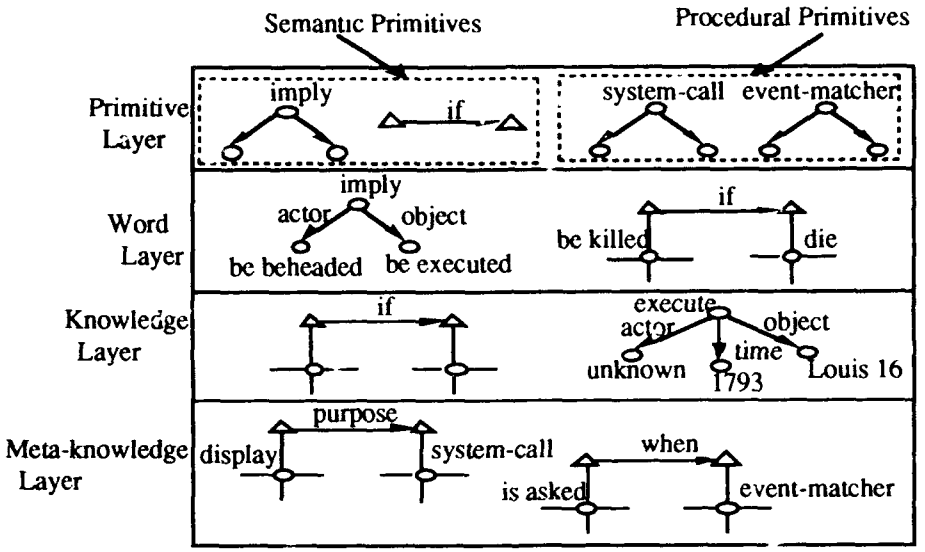


Fig. 2 Illusion of Knowledge Hierarchy

However, it is easy to see that the categorization is meaningless from the viewpoint of knowledge representation because there is no difference among the knowledge representations. All knowledge is represented in the case-based structure. Let's consider the sentence, "If a user asks a question, the system must answer the question". The sentence should be treated as meta-knowledge for reacting to the user's query and should be treated as simple factual knowledge for answering the question, "What do you do if a user asks a question?". Therefore, there are no differences among the various types of knowledge. This characteristic causes a problem in inference control.

In the conventional KE tool, meta-knowledge is represented in a special form and is processed by a supervisor inference unit. This separation of ordinal knowledge and meta-knowledge enables the system to control the system's behavior by using the meta-knowledge prior to the ordinal knowledge. However, the mechanism cannot be adopted because all knowledge in a text base is uniformly represented. Even if an item of knowledge looks like ordinal knowledge (e.g., domain-dependent know-how), the knowledge may be used as meta-knowledge by the fuzzy CBR. Since the fuzzy CBR tends to interpret the meaning of the sentence widely, the applicable range of the sentence is greatly extended.

We designed a new inference mechanism which wonders "What should I do?" every time an inference proceeds one step. All knowledge is used as meta-knowledge in the wondering phase. For example, the knowledge "if a user asks a question, the system must answer the question" is used as meta-knowledge during the wondering "what should I do?" that follows the user's inquiry. The truly essential points are to embed the meaning of "should", "I", and "do" as semantic or procedural primitives and to evoke the wondering every time an inference proceeds one step.

4.2 Fuzzy-centered Architecture (Positive Use of Fuzziness)

One of the inherent characteristics of natural language is fuzziness. This fuzziness varies from the ambiguity that exists in multiple parse trees for one sentence and multiple meanings for one word to the fuzziness of the meaning itself. Disambiguation algorithms have been developed to cope with the former type of fuzziness, but the latter type has not been considered in studies of natural language understanding systems.

The basic concept of fuzzy-centered architecture (positive use of fuzziness) is not a concrete architecture but a philosophy. It insists that fuzziness is not a bad feature of natural language but rather a very good feature. In disambiguation, fuzziness is handled as something bad, but disambiguation is not the only way to deal with the fuzziness of natural language. In natural language, fuzziness often has an important meaning, so in a truly human-friendly system, fuzziness cannot be ignored.

There are many area which fuzziness can play an important role in natural language processing. We focused on problem solving and learning which make best use of the fuzziness in natural language, as the first attempt toward implementing the fuzzy-centered architecture.

(1) Fuzziness of Meaning in Natural Language

What kind of fuzziness is there in natural language? Examples are fuzzy predicates (e.g., "tall", "old"), fuzzy modifiers (e.g., "very" , "more or less"), fuzzy modality (e.g., "most of", "usually"), and fuzzy inference (e.g., generalized modus ponens[12], gradual rules[2]).

Fuzziness can be roughly categorized as shown in the table in Fig. 3. Basically, fuzziness can be divided into ambiguity and fuzziness of meaning. For example, the ambiguous word "execute" can mean "start a program" or "kill a person". A sentence is ambiguous when it has multiple parse trees. For example, "A girl saw the boy with a telescope" can be interpreted in two ways, depending on who has the telescope.

	Word	**Sentence**
Ambiguity	Multi-meaning	Multi-structure
Fuzziness of Meaning **Simple**	"tall"	"the more, the more"
	well-structured universe of discourse	
Complex	"handsome"	proverbs
	unknown universe of discourse	

Fig. 3 Classification of fuzziness

Fuzziness in the meaning of a word can be classified as simple or complex. Simple fuzziness can be represented with a well-structured universe of discourse. For example, "tall" can be defined on a height axis. The universe of discourse for "height" is the continuous real number system. On the other hand, it's quite difficult to define such a universe of discourse for a word like "handsome".

Similarly, fuzziness in the meaning of a sentence can also be classified as simple or complex. An example of a simple fuzzy sentence is "the more xxx, the more yyy". In this sentence, the relationship between xxx and yyy is fuzzy. In other words, "the more ... the more ..." is a fuzzy sentence structure. A more complex sentence, such as a proverb, exhibits complex fuzziness.

The table in Fig. 4 shows the corresponding fuzzy theories that can be applied to the types of fuzziness listed in Fig. 3. For example, the simple fuzziness of a word can be processed by fuzzy set or fuzzy symbol theory. Generalized modus ponens[12] and gradual inference[2] are applicable for the simple fuzziness of a sentence. We can see that fuzzy theory provides us with ample methods for coping with the various types of fuzziness found in natural language.

	Word	**Sentence**
Ambiguity	disambiguation by world knowledge and user interaction	
Fuzziness of Meaning Simple	Fuzzy Set Fuzzy Symbol	Generalized Modus Ponens Gradual Inference
Complex	Fuzzy Rule Fuzzy Inference	(Fuzzy CBR)

Fig. 4 Application of fuzzy theory

(2) 3-Layered Fuzzy Inference Mechanism

Figure 5 shows our 3-layered fuzzy inference architecture for dealing with the fuzziness in Figs. 3 and 4 as a constituent of the fuzzy-centered architecture. In this 3-layered hierarchy of fuzzy inference, the basic inference mechanism is ordinary (non-fuzzy) inference, that is regular modus ponens. In this base layer, text symbols are treated simply as labels.

The second layer is fuzzy inference on a fuzzy set or fuzzy symbol. In this layer, a symbol is treated as either a fuzzy set (i.e., a sort of distribution over a universe of discourse) or a fuzzy symbol (i.e. words that can be calculated). A typical example for this level is deduction for result "the apple is very ripe" from the gradual rule "the more an apple is red, the more the apple is ripe." and the information "the apple is very red." Since this can be formalizes as $(A \rightarrow B, A') \Rightarrow B'$, where A' and B' are similar to A and B, this inference is a type of case-based reasoning. Current CBR methodology gives only a vague outline of the inference method, but the generalized modus ponens in fuzzy theory provides a concrete algorithm for executing a type of CBR having the formalism $(A \rightarrow B, A') \Rightarrow B'$. The top layer is fuzzy case-based reasoning. It is CBR extended by fuzzy theory, which tries to match two cases (which are also represented as text) on the basis of their fuzzy relationship. This can be formalized as $(A \rightarrow B, C) \Rightarrow D$, where C and D are different from A and B. In the problem solving phase, this mechanism is used in bottom to top manner, i.e. ordinary inference, fuzzy inference and fuzzy CBR are evoked in order. The learning phase, on the contrary, is a top to bottom process. This fuzzy CBR can be applied to the control of conversation by treating the conversation history as case data.

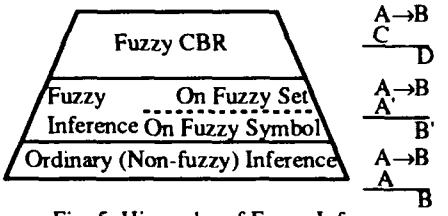


Fig. 5 Hierarchy of Fuzzy Inference

5. Two Experimental Systems

Here's a very short summary of the experimental systems. See [10,11] for detail.

5.1 A Text-based Natural Language System : AB-System[10]

An experimental text-based system, called the AB-System, was developed at Carnegie Mellon University to demonstrate the feasibility of using a text-based architecture.

System Architecture

Figure 6 shows the system architecture. The Generalized LR Parser/Compiler Version 8.4 and Generation Kit were used to implement the NL Parser and Generator. Common Lisp was used to implement the pattern matcher. The AB-System is currently running on an IBM-RT.

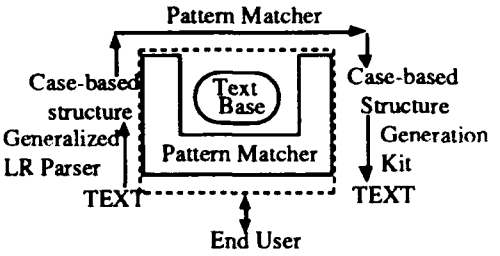


Fig. 6 AB-System Architecture

Knowledge Representation

The AB-System uses only eight cases: Actor, Purpose, Object, Cause, Instrument, Location, Time, and Unknown, but, "Event-Modify" and "Describe" are added to represent complex meanings.

Pattern Matcher

The AB-System only knows about semantic primitives and procedural primitives. The number of these primitives is quite limited, so the AB-System derives most of its power from the text base and the pattern matcher. The key part of the pattern matcher is the event matcher, which tries to unify two events.

In a pattern matcher, a new goal is sometimes generated for backward reasoning although the basic inference mechanism is forward reasoning (propagation of an input text in the text base network). The generated goal is represented in a case-based structure in order to handle this goal in the same way as a user inquiry. Moreover, all knowledge, including meta-knowledge, is represented in a case-based structure, so the status of network inference is represented as a case-based structure. This means that the system can always explain its own activity to provide the text generator with the inference status represented in the case-based structure.

Example of Communication

AB-system works in a mixed situation, i.e., QA about the French Revolution, knowledge debugging and computer front-end. When a user inputs a text, the AB-system transforms it into a case-based structure and checks whether or not the knowledge is already known. If the knowledge is unknown, it stores the text in a text base, transforms it back to natural language and displays "Okay. I understood." followed by the generated text. Examples of the inputs are listed below.

"The French Revolution occurred in 1789 in France."
"Robespierre suspended the constitution and assumed the dictatorial powers."
"The article describes the person, if a person is described in a article."
"AB system-calls vi with the file in order to display a file, if the file is long."
"When was the `Queen Antoinette' executed, if `to be beheaded' means `to be executed' ?"
"Show all knowledge about `Queen Antoinette'."
"Display the article that describes `the person who was executed in 1793'."

Fig. 7 Examples of the inputs

5.2 A Fuzzy Expert System: FOREX [4,11]

We have built a fuzzy expert system that predicts the exchange rate trends of the yen against the dollar based on 5000 fuzzy rules and 300 fuzzy frames. It makes full use of fuzzy theory, i.e. fuzzy set theory, fuzzy logic, fuzzy measures and fuzzy integrals[6].

Overview of FOREX

FOREX consists of a state recognition part and a scenario evaluation part. First, numerical data and news data are entered into the state recognition part and each data item is converted into one or a collection of fuzzy qualitative linguistic values. Several important indexes which strongly influence the foreign exchange market are deduced as inputs to the scenario evaluation part. Second, the important indexes are used to select the most suitable scenario that describes possible future changes in the exchange rate. The special features are:

Separation of State Recognition Part and Scenario Evaluation Part

In the state recognition part, numerical data and news are analyzed and synthesized into fuzzy qualitative indicators. Then, in the scenario evaluation part, the fuzzy qualitative indicators are used to choose the most likely scenario stored in FOREX.

Four-layered State Recognition

The state recognition part is divided into four levels, 0 to 3, the highest of which presents the highest degree of abstraction. Raw numerical data in level 0 is evaluated and translated into fuzzy data. Level 2 looks like a state transition network in which each state stands for an aspect of economic fundamentals. News is given in level 2 directly. The state of the network is summarized into a few dozen indicators in level 3 for use by the scenario evaluation part. The state recognition part corresponds to the middle part of 3-layered fuzzy inference mechanism.

State Representation by 3x3 Fuzzy Linguistic Qualitative Values

All states in levels 1 to 3 are represented by 3x3 linguistic fuzzy variables, i.e. combinations of past/current/future and level/differential/quadratic differentials.

Scenario Evaluation by Fuzzy Integral

The conditions of the scenario are given as fuzzy measures on the state values in level 3. Each scenario is evaluated by fuzzy integrals and sorted by fuzzy ranking.

Treatment of Fuzziness

Although FOREX uses various fuzzy methods, we found that the most effective methods are fuzzification of fuzzy "words" and generalized modus ponens, especially gradual inference.

Fuzzification of Words

Most of the state values that need to be represented in FOREX are mainly of the type summed up by the statement: "Roughly somewhat about here, but I'm not sure exactly where". Such data are transformed into possibility distributions (fuzzy set). For example, "US-short-term-interest is somewhat high" can be represented by a triangular possibility distribution with a value of one on the linguistic label "high" and a value of less than one on other labels ("very high", "more-or-less high", and so on). All linguistic statements and numerical data are converted to possibility distributions over the linguistic label. This transformation reflects the feeling that the word "high" does not mean "strict high" but "vague high".

Generalized Modus Ponens

One of the most important features of fuzzy logic is that it can represent the fuzziness in the relation between a condition and a consequent. When an expert says "If FF-rate is high then US-short-term-interest is high", it is often equivalent to "The more FF-rate is high , the more US-short-term-interest is high". That is the gradual nature of the rule structure.

6. Summary

We proposed two new design concepts, text-based architecture and fuzzy-centered architecture (positive use of fuzziness), to overcome problems and limitations of conventional natural language understanding systems, including (1) the closed nature of systems, (2) the fact that implicit and inherent knowledge is represented in a special form, (3) deep structure representation of meaning, and (4) neglect of the fuzziness of meaning.

The text-based architecture was directly derived from problems (1), (2) and (3). The four subconcepts, (1) combination of case-based structure and text base, (2) semantic and procedural primitives, (3) network inference, and (4) non-hierarchical wondering inference, are proposed to make a text-based architecture feasible.

The fuzzy-centered architecture is derived from the fact that most natural language understanding systems can only cope with ambiguity, even though the fuzziness of the meaning itself is quite important. The 3-layered hierarchy of fuzzy inference was presented as a possible inference mechanism. The hierarchy consists of (1) ordinary (non-fuzzy) inference, (2) fuzzy inference on fuzzy sets or fuzzy symbols, and (3) fuzzy case-based reasoning.

Two experimental systems, the AB-System and FOREX, demonstrate the feasibility of new design concepts, although they do not cover the full range of the new concepts. We are now doing fundamental research on the problems discussed above to fully implement FLINS based on the two new design concepts.

References

1. C. Dimacro and G. Hirt: Current Approaches To Natural Language Semantics, Tutorial Proceedings, AAAI-90 (1990).
2. D. Dubois and H. Prade: Gradual Inference Rules in Approximate Reasoning, Information Science, Vol.61, pp.103-122 (1992).
3. C.J Fillmore: The Case for Case, in Bach, E. and Harms, R.T. (eds.): Universals in Linguistics, Holt Rinehart and Winston New York, pp. 1-90 (1968).
4. Y. Katoh, H. Yuize, M. Yoneda, K. Takahashi, S. Tano, T. Yagyu, M. Grabish and S. Fukami: Gradual Rules in a Decision Support System for Foreign Exchange Trading, Int'l Conf. on Fuzzy Logic & Neural Net., pp.625-628 (1992).
5. J.M. L ine : PRAGMA - A Flexible Bidirectional Dialogue System, AAAI-90, pp.964-969 (1990).
6. T. Murofushi and M. Sugeno: An Interpretation of Fuzzy Measures and Choquet Integral with resprect to a Fuzzy Measure, Fuzzy Sets and Systems, Vol.29, pp.201-227 (1989).
7. S. Nirenburg Machine Translation, Studies in Natural Language Processing, Cambridge University Press (1987).
8. R.C. Shank: Conceptual Dependency: A Theory of Natural Language Analysis, Cognitive Psychology, Vol. 3 (1972).
9. O. Stock: Natural Language and Exploration of an Information Space: the ALFresco Interactive System, IJCAI-91, pp.972-978 (1991).
10. S. Tano: Natural Language Understanding System using Text-based Inference, CMU-CMT-91-124 (1991).
11. S. Tano, H. Yuize, T. Yagyu, M. Yoneda, Y. Katoh, M. Grabish and S. Fukami: FOREX: Foreign Exchange Trade Support Expert System, International Fuzzy Engineering Symposium '91, pp.1114-1115 (1991).
12. R.R. Yager: FUZZY SETS AND APPLICATIONS: Selected Papers by L.A. Zadeh, JOHN WILEY & SONS (1988).

Boolean Reasoning for Decision Rules Generation

Andrzej Skowron

Institute of Mathematics, University of Warsaw
02-097 Warsaw, Banacha 2, Poland
e-mail: skowron@mimuw.edu.pl

Abstract. In the paper we investigate the generation problem of optimal decision rules with some certainty coefficients based on belief [7] and rough membership functions [6]. We show that the problems of optimal rules generation can be solved by boolean reasoning [2].

1 Introduction

In the paper we discuss the problem of optimal decision rules generation. Decision rules have the following form: $\tau \Rightarrow \tau'$ where τ, τ' are boolean combinations of descriptors built from conditions and a decision approximating the expert decision [9], respectively. The decision rules are generated with some certainty coefficients expressed by the basic functions of evidence theory (basic probability assignments, belief and plausibility functions) and rough membership functions computable from a given decision table. These coefficients can be used in the decision making. The method of rules generation is based on a construction of appropriate boolean functions from modified discernibility matrices [10]. The optimal forms of the rules with respect to the number of attributes occurring in them are obtained from prime implicants of those functions. Two kinds of optimal decision rules are considered: locally optimal rules and globally optimal rules. We show that our method can be applied also for construction of decision rules based on β-lower and β-upper approximations of sets [12].

2 Information Systems and Rough Sets

Information systems [4,5] are used for representing knowledge suitable for appropriate decision making.

Rough sets have been introduced [4,5] as a tool to deal with inexact, uncertain or vague knowledge in artificial intelligence applications, like for example knowledge based systems in medicine, natural language processing, pattern recognition, decision systems, approximate reasoning.

In this section we recall some basic notions related to information systems and rough sets.

An *information system* is a pair $\mathbb{A} = (U, A)$, where U is a non-empty, finite set called the *universe* and A – a non-empty, finite set of *attributes*, i.e. $a : U \rightarrow V_a$ for $a \in A$, where V_a is called the *value set* of a.

Elements of U are called *objects* and interpreted as, e.g. cases, states, processes, patients, observations. Attributes are interpreted as features, variables, characteristic conditions etc.

We consider a special case of information systems called decision tables. A *decision table* is any information system of the form $\mathbb{A} = (U, A \cup \{d\})$, where $d \notin A$ is a distinguished attribute called *decision* . The elements of A are called *conditions*.

It is enough to consider decision tables with one decision because always by simple coding one can transform any decision table with more than one decision into a decision table with exactly one decision. One can interpret a decision attribute as a kind of classification of the universe of objects given by an expert decision-maker, operator, physician, etc.

The cardinality of the image $\Theta_{\mathbb{A}} = \{k : d(s) = k \text{ for some } s \in U\}$ is called the *rank* of d and is denoted by $r(d)$. We assume, without loss of the generality, that the set V_d of values of the decision d is equal to $\{1, \ldots, r(d)\}$.

Let us observe that the decision d determines the partition $CLASS_{\mathbb{A}}(d) = \{X_1, \ldots, X_{r(d)}\}$ of the universe U, where $X_k = d^{-1}(\{k\})$ for $1 \leq k \leq r(d)$. $CLASS_{\mathbb{A}}(d)$ will be called the *classification of objects in* $\mathbb{A}$ *determined by the decision* d.

Let $\mathbb{A} = (U, A)$ be an information system. With every subset of attributes $B \subseteq A$, an equivalence relation, denoted by $IND_{\mathbb{A}}(B)$ (or $IND(B)$) called the *B-indiscernibility relation*, is associated and defined as follows:

$$IND(B) = \{(s, s') \in U^2 : \text{ for every } a \in B, a(s) = a(s')\}$$

Objects s, s' satisfying the relation $IND(B)$ are indiscernible by attributes from B.

Let $\mathbb{A}$ be an information system with n objects. By $M(\mathbb{A})$ [10] we denote an $n \times n$ matrix (c_{ij}), called the *discernibility matrix* of $\mathbb{A}$ such that

$$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\} \text{ for } i, j = 1, \ldots, n.$$

A discernibility *function* $f_{\mathbb{A}}$ for an information system $\mathbb{A}$ is a boolean function of m boolean variables $\overline{a}_1, \ldots, \overline{a}_m$ corresponding to the attributes $a_1, \ldots, a_m$ respectively, and defined as follows:

$$f_{\mathbb{A}}(\overline{a}_1, \ldots, \overline{a}_m) = \bigwedge \{\bigvee \overline{c}_{ij} : 1 \leq j < i \leq n, c_{ij} \neq \emptyset\}$$

where $\overline{c}_{ij} = \{\overline{a} : a \in c_{ij}\}$

It can be shown [10] that the set of all *prime implicants* of $f_{\mathbb{A}}$ determines the set of all *reducts* of $\mathbb{A}$. Here we apply an analogous method for the decision rules generation by a generalization of the discernibility matrix notion. The modified discernibility matrix $MG(\mathbb{A})$ is a subset of $\mathbb{P}(\mathbb{A}) \times \{1, \ldots, n\} \times \{1, \ldots, n\}$ computable from $M(\mathbb{A})$ (where by $\mathbb{P}(A)$ we denote the power set of A). By $f_{MG(A)}$ we denote a boolean function constructed from $MG(\mathbb{A})$ in an analogous way as $f_{\mathbb{A}}$ from $M(\mathbb{A})$. Different forms of decision rules are obtained from different constructions of sets $MG(\mathbb{A})$ (see Section 4).

If $\mathbb{A} = (U, A)$ is an information system, $B \subseteq A$ is a set of attributes and $X \subseteq U$ is a set of objects then the sets

$$\{s \in U : [s]_B \subseteq X\} \quad \text{and} \quad \{s \in U : [s]_B \cap X \neq \emptyset\}$$

are called *B-lower* and *B-upper approximation* of X in $\mathbb{A}$, and they are denoted by $\underline{B}X$ and $\overline{B}X$, respectively.

The set $BN_B(X) = \overline{B}X - \underline{B}X$ will be called the *B-boundary* of X. When $B = A$ we write also $BN_{\mathbb{A}}(X)$ instead of $BN_A(X)$.

Sets which are unions of some classes of the indiscernibility relation $IND(B)$ are called *definable* by B. Some subsets (categories) of objects in an information system cannot be expressed exactly by employing available attributes but they can be roughly defined. The set X is B-definable if $\overline{B}X = \underline{B}X$.

The set $\underline{B}X$ is the set of all elements of U which can be with certainty classified as elements of X, having the knowledge represented by attributes from B; $\overline{B}X$ is the set of elements of U which can be classified as elements possibly belonging to X, employing the knowledge represented by attributes from B; set $BN_B(X)$ is the set of elements which cannot be classified either to X or to $-X$ having knowledge B.

Every information system $\mathbb{A} = (U, A)$ determines an *information function*

$$Inf_{\mathbb{A}} : U \to \mathbb{P}(A \times \bigcup_{a \in A} V_a)$$

defined as: $Inf_{\mathbb{A}}(x) = \{(a, a(x)) : a \in A\}$. The set $\{Inf_{\mathbb{A}}(x) : x \in U\}$ is called the $\mathbb{A}$-*information set* and it is denoted by $INF(\mathbb{A})$.

Let $\mathbb{A} = (U, A)$ be an information system and let $\emptyset \neq X \subseteq U$. The *rough* $\mathbb{A}$-*membership function of the set* X (or *rm-function*, for short), denoted by $\mu_X^{\mathbb{A}}$, is defined as:

$$\mu_X^{\mathbb{A}}(x) = \frac{|[x]_A \cap X|}{|[x]_A|}, \quad \text{for} \quad x \in U$$

One can observe that this is a generalization of the set characteristic function. Properties of rm-functions are discussed in [6].

For every $X \subseteq U$ the rough $\mathbb{A}$-information function, $\hat{\mu}_X^{\mathbb{A}}$ is defined by:

$$\hat{\mu}_X^{\mathbb{A}}(u) = \mu_X^{\mathbb{A}}(x), \quad \text{where} \quad u \in INF(\mathbb{A}) \quad \text{and} \quad Inf_{\mathbb{A}}(x) = u$$

3 Rough Set Interpretation of the Basic Functions of the Evidence Theory

The classification problems are central for the rough set approach [5] as well as for the evidence theoretic approach [7].

The fundamental assumption in the rough set approach is the following one: the objects from the universe are perceived only through the accessible information about them, i.e. the values of attributes which can be evaluated on these objects. Objects with the same information are indiscernible. In the consequence

the classification of objects is based on the accessible information about them, not on objects themselves. Together with the information about objects from a finite set of objects the classification of them delivered by an expert is given. The classification problem in this case is related to the question in what extent it is possible to reflect by accessible (condition) attributes the classification done by expert.

In the evidence theory [7] the information about sets creating a partition is embedded directly in some numerical functions whereas in the case of rough set approach the information about classified sets and objects is included in a decision table. The evidence theory approach is based on the idea of placing a number from the interval [0, 1], given e.g. by an expert, to indicate a degree of belief for a given proposition on the basis of a given evidence.

We show [9] how to compute the basic functions of evidence theory from a given decision table. First we recall some basic notions of evidence theory [7].

A *frame of discernment* Θ is a finite non-empty set.

The *basic probability assignment* (bpa) on Θ is any function $m : \mathbb{P}(\Theta) \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative reals, satisfying the following two conditions:

$$m(\emptyset) = 0 \quad \text{and} \quad \sum_{\Delta \subseteq \Theta} m(\Delta) = 1$$

For a given bpa m two functions are defined.

A function $Bel : \mathbb{P}(\Theta) \to \mathbb{R}_+$ is called the *belief function* over Θ (generated by m) iff for any $\theta \subseteq \Theta$

$$Bel(\theta) = \sum_{\Delta \subseteq \theta} m(\Delta)$$

A function $Pl : \mathbb{P}(\Theta) \to \mathbb{R}_+$ is called the *plausibility function* over Θ (generated by m) iff for any $\theta \subseteq \Theta$

$$Pl(\theta) = \sum_{\Delta \cap \theta \neq \emptyset} m(\Delta)$$

Let us observe that the inequality $Bel(\theta) + Bel(\Theta - \theta) \leq 1$ can not be in general reduced to the equality $Bel(\theta) + Bel(\Theta - \theta) = 1$ ($P(\theta) + P(\Theta - \theta) = 1$ for any probability function on $\mathbb{P}(\Theta)$). This allows to take into account ignorance, e.g. if we have no evidence at all, for or against θ then $Bel(\theta) = Bel(\Theta - \theta) = 0$.

The plausibility function Pl is definable by the belief function, namely: $Pl(\theta) = 1 - Bel(\Theta - \theta)$ for $\theta \subseteq \Theta$.

Now we will describe how these functions can be defined and interpreted on a basis of rough set approach. For a given decision table one can define a new objects classification approximating the classification given by the decision attribute. The approximation is constructed on the basis of conditions in decision table.

The set $\Theta_{\mathbb{A}} = \{1, \ldots, r(d)\}$ determined by the decision d in $\mathbb{A} = (U, A \cup \{d\})$ is called the *frame of discernment defined by d in* $\mathbb{A}$.

We say that the frame of discernment Θ is *compatible* with $\mathbb{A}$ if $r(d) = |\Theta|$. If $\Theta = \{\theta_1, \ldots, \theta_k\}$ is compatible with $\Theta_{\mathbb{A}}$ then χ denotes the bijection between

Θ and $\Theta_{\mathbb{A}}$ defined by: $\chi(\theta_i) = i$ for $i = 1, \ldots, k$ extended on subsets of Θ by $\chi(\theta) = \{i : \theta_i \in \theta\}$, where $\theta \subseteq \Theta$.

The objects from the universe U of $\mathbb{A}$ can be classified employing the knowledge represented by conditions from A. This allows to decide that either an object belongs into the lower approximation of a given set $X \subseteq U$ or it is in the complement of the upper approximation of X or belongs to the boundary region corresponding to X. Moreover, one can, in some sense, better classify objects from the boundary regions. This is based on the following observation [9]:

Proposition 3.1 *Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table. The family of all non-empty sets from*

$$\{\underline{A}X_1, \ldots, \underline{A}X_{r(d)}\} \cup \{Bd_{\mathbb{A}}(\theta) : \theta \subseteq \Theta_{\mathbb{A}} \text{ and } |\theta| > 1\}$$

(where $Bd_{\mathbb{A}}(\theta) = \bigcap_{i \in \theta} BN_A(X_i) \cap \bigcap_{i \notin \theta} -BN_A(X_i)$) is a partition of the universe U. Moreover the following equality holds:

$$\bigcup_{i \in \theta} \underline{A}X_i \cup \bigcup_{\Delta \subseteq \theta, |\Delta| > 1} Bd_{\mathbb{A}}(\Delta) = \underline{A} \bigcup_{i \in \theta} X_i \text{ for } \theta \subseteq \Theta_{\mathbb{A}} \text{ with } |\theta| > 1 .$$

□

The classification (partition) of the universe U described in Proposition 3.1 is called the *standard classification of U approximating in $\mathbb{A}$ the classification* $CLASS_{\mathbb{A}}(d)$ (given by expert).

By $APP_CLASS_{\mathbb{A}}(d)$ we denote the family:

$$\{\underline{A}X_1, \ldots, \underline{A}X_{r(d)}\} \cup \{Bd_{\mathbb{A}}(\theta) : \theta \subseteq \Theta_{\mathbb{A}} \text{ and } |\theta| > 1\}$$

We have a clear interpretation of the new classification. An object from the universe U of $\mathbb{A}$ is represented by an information (from $INF(\mathbb{A})$) described in the rows of $\mathbb{A}$. This object can be classified exactly on the basis of that information only when the category (i.e. the equivalence class of the indiscernibility relation $IND_{\mathbb{A}}(A)$) corresponding to that information is included in X_i for some i. Otherwise, that category is included in a boundary region of the form $Bd_{\mathbb{A}}(\theta)$, for some θ. Then the considered object from U represented by a given information can be classified to the boundary region of all sets X_i , where $i \in \theta$ (i.e. it can be classified into the union $\bigcup_{i \in \theta} X_i$ but there is no enough information to decide either in which of the sets X_i it is or to eliminate some hypotheses from $\{X_i : i \in \theta\}$).

There is a natural correspondence between subsets of $\Theta_{\mathbb{A}}$ and elements of $APP_CLASS_{\mathbb{A}}(d)$, which can be expressed by the following function:

$$F_{\mathbb{A}}(\theta) = \begin{cases} \underline{A}X_i & \text{if } \theta = \{i\} \text{ for some } i \ (1 \leq i \leq r(d)) \\ \emptyset & \text{if } \theta = \emptyset \\ Bd_{\mathbb{A}}(\theta) & \text{if } |\theta| > 1 \end{cases}$$

Now we can define the injection $\partial_{\mathbb{A}} : U \to \mathbb{P}(\Theta_{\mathbb{A}})$ such that $\partial_{\mathbb{A}}(s)$ is the unique subset θ of $\Theta_{\mathbb{A}}$ such that $s \in F_{\mathbb{A}}(\theta)$. The function $\partial_{\mathbb{A}}$ can be treated as a new decision attribute (defined by conditions in $\mathbb{A}$) approximating the decision d.

Let $\Theta = \{\theta_1, \ldots, \theta_k\}$ be a frame of discernment compatible with $\mathbb{A}$ and let $\chi : \Theta \to \Theta_{\mathbb{A}}$ be the standard bijection between Θ and $\Theta_{\mathbb{A}}$ i.e. $\chi(\theta_i) = i$ for $i = 1, \ldots, k$. The function $m_{\mathbb{A}} : \mathbb{P}(\Theta) \to R_+$, called the *standard basic probability assignment* (defined by Θ, $\mathbb{A}$ and χ) is defined as :

$$m_{\mathbb{A}}(\theta) = \frac{|F_{\mathbb{A}}(\chi(\theta))|}{|U|}, \quad \text{for any} \quad \theta \subseteq \Theta .$$

Proposition 3.2 [9] *The function $m_{\mathbb{A}}$ defined above is a basic probability assignment (in the sense of evidence theory).*

□

The *belief function* $Bel_{\mathbb{A}}$ for the frame of discernment Θ and decision table $\mathbb{A}$ compatible with Θ is defined as:

$$Bel_{\mathbb{A}}(\theta) = \sum_{\Delta \subseteq \theta} m_{\mathbb{A}}(\Delta)$$

where $\theta \subseteq \Theta$ and $m_{\mathbb{A}}$ is the standard probability assignment for Θ and $\mathbb{A}$.

Theorem 3.3 *Let Θ be a frame of discernment compatible with the decision table $\mathbb{A} = (U, A \cup \{d\})$ and let χ be the standard bijection between Θ and $\Theta_{\mathbb{A}}$. For any $\theta \subseteq \Theta$ the following equality holds:*

$$Bel_{\mathbb{A}}(\theta) = \frac{|\underline{A} \bigcup_{i \in \chi(\theta)} X_i|}{|U|}$$

□

The belief function $Bel_{\mathbb{A}}$ is Bayesian iff all sets from $CLASS_{\mathbb{A}}(d)$ are definable by the set A of conditions. In particular the belief function $Bel_{\mathbb{A}'}$, where $\mathbb{A}' = (U, A \cup \{\partial_{\mathbb{A}}\})$, is Bayesian. □

Corollary 3.4 *Let Θ be a frame of discernment compatible with the decision table $\mathbb{A} = (U, A \cup \{d\})$. For any $\theta \subseteq \Theta$ the following equality holds:*

$$Pl_{\mathbb{A}}(\theta) = \frac{|\overline{A} \bigcup_{i \in \chi(\theta)} X_i|}{|U|}$$

□

4 Decision Rules Generation

In this section we investigate different types of decision rules (for a given decision table) with some certainty coefficients. These coefficients are computable from a given decision table and can be used in the decision making.

Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table and let $V = \bigcup_{a \in A} V_a \cup V_d$.

The atomic formulas over $B \subseteq A \cup \{d\}$ and V are expressions of the form $a = v$, called *descriptors* over B, where $a \in B$ and $v \in V_a$. The set $\mathbb{F}(B, V)$ of formulas over B is the least set containing all atomic formulas over B and closed with respect to the classical propositional connectives $\vee$ (disjunction), $\wedge$ (conjunction) and $\neg$ (negation).

Let $\tau \in \mathbb{F}(B, V)$ (where $B \subseteq A \cup \{d\}$) then by $\tau_{\mathbb{A}}$ we denote the meaning of τ in the decision table $\mathbb{A}$, i.e. the set of all objects in U with property τ, defined inductively as follows:

1. if τ is of the form $a = v$ then $\tau_{\mathbb{A}} = \{s \in U : a(s) = v\}$;
2. $(\tau \wedge \tau')_{\mathbb{A}} = \tau_{\mathbb{A}} \cap \tau'_{\mathbb{A}}$; $(\tau \vee \tau')_{\mathbb{A}} = \tau_{\mathbb{A}} \cup \tau'_{\mathbb{A}}$; $(\neg\tau)_{\mathbb{A}} = U - \tau_{\mathbb{A}}$.

The set $\mathbb{F}(A, V)$ is called the set of condition formulas in $\mathbb{A}$ and is denoted by $\mathbb{C}_{\mathbb{A}}$. The set $\mathbb{F}(\{\partial_{\mathbb{A}}\}, V')$, where $V' = (V - V_d) \cup V_{\partial_{\mathbb{A}}}$ is called the set of decision formulas in $\mathbb{A}^* = (U, A \cup \{\partial_{\mathbb{A}}\})$ and is denoted by $\mathbb{D}_{\mathbb{A}}$.

If $u = \{(a = v), \ldots, (a_r = v_r)\}$ then by τ_u we denote the conjunction $(a_1 = v_1) \wedge \ldots \wedge (a_r = v_r)$. By $A(\tau)$ we denote the set of attributes occurring in the formula τ or corresponding to boolean variables occurring in the formula τ

A *decision rule* for $\mathbb{A}$ is any expression of the form

$$\tau \Rightarrow \tau' \quad \text{where} \quad \tau \in \mathbb{C}_{\mathbb{A}} \quad \text{and} \quad \tau' \in \mathbb{D}_{\mathbb{A}}.$$

The decision rule $\tau \Rightarrow \tau'$ for $\mathbb{A}$ is *true in* $\mathbb{A}$ iff $\tau_{\mathbb{A}} \subseteq \tau_{\mathbb{A}^*}$; if $\tau_{\mathbb{A}} = \tau_{\mathbb{A}^*}$ then we say that the rule is $\mathbb{A}$-*exact*.

We are looking for two optimal forms of those rules with respect to the number of attributes on the left hand side. The $\mathbb{A}$-exact rule $\tau \Rightarrow \tau'$ is $\mathbb{A}$-*locally optimal* iff there is no $\mathbb{A}$-exact rule $\tau'' \Rightarrow \tau'$ such that $A(\tau'') \subset A(\tau)$. The $\mathbb{A}$-exact rule $\tau \Rightarrow \tau'$ is $\mathbb{A}$ -*globally optimal* iff $|A(\tau)| \leq |A(\tau'')|$ for any $\mathbb{A}$-exact rule $\tau'' \Rightarrow \tau'$.

The optimal rules we obtain applying the mentioned in Section 2 method based on the modified discernibility matrices.

Decision rules of the first type have the following form:

$$\tau \Rightarrow \partial_{\mathbb{A}} = \Delta, \quad \text{where} \quad \Delta \subseteq \{1, \ldots, r(d)\} \quad \text{and} \quad \tau \in \mathbb{C}_{\mathbb{A}}$$

One can prove the following theorem:

Theorem 4.1 *Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table, $\Delta \subseteq \{1, \ldots, r(d)\}$ and let the modified discernibility matrix $MG_1(\mathbb{A})$ (for a given Δ) be equal to*

$$\{c_{ij} - \{d\} : c_{ij} \in M(\mathbb{A}) \ \& \ (\partial_{\mathbb{A}}(x_i) = \Delta \text{ xor } \partial_{\mathbb{A}}(x_j) = \Delta)\}$$

Then for any prime implicant t of $f_{MG_1(\mathbb{A})}$ we have:

1. $(\partial_{\mathbb{A}} = \Delta)_{\mathbb{A}^*} = (\bigvee\{\tau_u : u \in INF_1(A(t), \Delta, \mathbb{A}^*)\})_{\mathbb{A}}$
   where $INF_1(A(t), \Delta, \mathbb{A}^*) = \{\{(a, a(x)) : a \in A(t)\} : \partial_{\mathbb{A}}(x) = \Delta \& x \in U\}$.
2. $\bigvee\{\tau_u : u \in INF_1(A(t), \Delta, \mathbb{A}^*)\} \Rightarrow \partial_{\mathbb{A}} = \Delta$ *is the* $\mathbb{A}$*-locally optimal decision rule. If the set* $A(t)$ *of attributes occurring in the prime implicant* t *has the minimal number of attributes (among all prime implicants of* $f_{MG_1(\mathbb{A})}$*) then the rule is* $\mathbb{A}$*-globally optimal.*
3. $m_{\mathbb{A}}(\chi^{-1}(\Delta)) = \dfrac{|\{x \in U : \partial_{\mathbb{A}}(x) = \Delta\}|}{|U|}$. □

Condition 1 states that the set defined by Δ in $\mathbb{A}^*$ is definable by the "trace" in $\mathbb{A}$ of any prime implicant of $f_{MG_1(\mathbb{A})}$ in $\mathbb{A}$. Hence the decision rule described in Condition 2 is true in $\mathbb{A}^*$. The rule describes information on the basis of which it is possible to classify objects into the union $\bigcup_{i\in\Delta} X_i$ without a possibility to eliminate any hypothesis from $\{X_i : i \in \Delta\}$. The value $m_{\mathbb{A}}(\chi^{-1}(\Delta))$ describes a "chance" that an object chosen from $\mathbb{A}$ is classified by $\partial_{\mathbb{A}}$ into Δ (Condition 3).

Decision rules of the second type have the following form:

$$\tau \Rightarrow \bigvee\{\partial_{\mathbb{A}} = \theta : \theta \subseteq \Delta\}, \quad \text{where} \quad \Delta \subseteq \{1, \ldots, r(d)\} \quad \text{and} \quad \tau \in \mathbb{C}_{\mathbb{A}}$$

One can prove the following theorem:

Theorem 4.2 *Let* $\mathbb{A} = (U, A \cup \{d\})$ *be a decision table,* $\Delta \subseteq \{1, \ldots, r(d)\}$ *and let the modified discernibility matrix* $MG_2(\mathbb{A})$ *(for a given* Δ*) be equal to*

$$\{c_{ij} - \{d\} : c_{ij} \in M(\mathbb{A}) \ \&\ (\partial_{\mathbb{A}}(x_i) \subseteq \Delta \text{ xor } \partial_{\mathbb{A}}(x_j) \subseteq \Delta)\}$$

Then for any prime implicant t *of* $f_{MG_2}(\mathbb{A})$ *we have:*

1. $(\bigvee\{\partial_{\mathbb{A}} = \theta : \theta \subseteq \Delta\})_{\mathbb{A}^*} = (\bigvee\{\tau_u : u \in INF_2(A(t), \Delta, \mathbb{A}^*)\})_{\mathbb{A}}$
   where $INF_2(A(t), \Delta, \mathbb{A}^*) = \{\{(a, a(x)) : a \in A(t)\} : \partial_{\mathbb{A}}(x) \subseteq \Delta \& x \in U\}$;
2. $\bigvee\{\tau_u : u \in INF_2(A(t), \Delta, \mathbb{A}^*)\} \Rightarrow \bigvee\{\partial_{\mathbb{A}} = \theta : \theta \subseteq \Delta\}$
   is the $\mathbb{A}$*-locally optimal decision rule. If the set* $A(t)$ *of attributes occurring in the prime implicant* t *has the minimal number of attributes (among all prime implicants of* $f_{MG_2(A)}$*) then the rule is* $\mathbb{A}$*-globally optimal.*
3. $Bel_{\mathbb{A}}(\chi^{-1}(\Delta)) = \dfrac{|\{x \in U : \partial_{\mathbb{A}}(x) \subseteq \Delta\}|}{|U|}$

□

Again Condition 1 states that the set defined by $\bigvee\{\partial_{\mathbb{A}} = \theta : \theta \subseteq \Delta\}$ in $\mathbb{A}^*$ is definable by the "trace" in $\mathbb{A}$ of any prime implicant of $f_{MG_2(\mathbb{A})}$ in $\mathbb{A}$. Hence the decision rule described in Condition 2 is true in $\mathbb{A}^*$. The rule describes conditions on the basis of which it is possible to classify objects with certainty into the union $\bigcup_{i\in\Delta} X_i$, i.e. to $\underline{A}\bigcup_{i\in\Delta} X_i$. The value $Bel_{\mathbb{A}}(\chi^{-1}(\Delta))$ describes a "chance" that an object chosen from $\mathbb{A}$ is classified with certainty into the union $\bigcup_{i\in\Delta} X_i$ (Condition 3 and Theorem 3.3).

Decision rules of the third type have the following form:

$$\tau \Rightarrow \bigvee\{\partial_{\mathbb{A}} = \theta : \theta \cap \Delta \neq \emptyset\}, \quad \text{where} \quad \Delta \subseteq \{1, \ldots, r(d)\} \quad \text{and} \quad \tau \in \mathbb{C}_{\mathbb{A}}.$$

One can prove the following theorem:

Theorem 4.3 *Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision table, $\Delta \subseteq \{1, \ldots, r(d)\}$ and let the modified discernibility matrix $MG_3(\mathbb{A})$ (for a given Δ) be equal*

$$\{c_{ij} - \{d\} : c_{ij} \in M(\mathbb{A}) \ \& \ (\partial_{\mathbb{A}}(x_i) \cap \Delta = \emptyset \ \text{ xor } \ \partial_{\mathbb{A}}(x_j) \cap \Delta = \emptyset)\}$$

Then for any prime implicant t of $f_{MG_3(\mathbb{A})}$ we have:

1. $(\bigvee\{\partial_{\mathbb{A}} = \theta : \theta \cap \Delta \neq \emptyset\})_{\mathbb{A}^*} = (\bigvee\{\tau_u : u \in INF_3(A(t), \Delta, \mathbb{A}^*)\})_{\mathbb{A}}$
   where $INF_3(A(t), \Delta, \mathbb{A}^*) = \{\{(a, a(x)) : a \in A(t)\} : \partial_{\mathbb{A}}(x) \cap \Delta \neq \emptyset \& x \in U\}$;
2. $\bigvee\{\tau_u : u \in INF_3(A(t), \Delta, \mathbb{A}^*)\} \Rightarrow \bigvee\{\partial_{\mathbb{A}} = \theta : \theta \cap \Delta \neq \emptyset\}$
   is the $\mathbb{A}$-locally optimal decision rule. If the set $A(t)$ of attributes occurring in the prime implicant t has the minimal number of attributes (among all prime implicants of $f_{MG_3(\mathbb{A})}$) then the rule is $\mathbb{A}$-globally optimal.
3. $Pl_{\mathbb{A}}(\chi^{-1}(\Delta)) = \dfrac{|\{x \in U : \partial_{\mathbb{A}}(x) \cap \Delta \neq \emptyset\}|}{|U|}$.

□

Again Condition 1 states that the set defined by $\bigvee\{\partial_{\mathbb{A}} = \theta : \theta \cap \Delta \neq \emptyset\}$ in $\mathbb{A}^*$ is definable by the "trace" in $\mathbb{A}$ of any prime implicant of $f_{MG_3(\mathbb{A})}$ in $\mathbb{A}$. Hence the decision rule described in Condition 2 is true in $\mathbb{A}^*$. The rule describes information about conditions from A on the basis of which one can classify objects as possibly belonging to $\bigcup_{i \in \Delta} X_i$. The value $Pl_{\mathbb{A}}(\chi^{-1}(\Delta))$ describes a "chance" that an object chosen from $\mathbb{A}$ is classified as possibly belonging to $\bigcup_{i \in \Delta} X_i$ (Condition 3 and Corollary 3.4).

From above facts follows that the certainty coefficients computed as the values of the basic probability assignments, belief or plausibility functions can be important in the decision making. The rough membership functions can be also used with the same purpose. For example, in the case of decision rules of first type a distribution of objects satisfying disjuncts τ_u (occurring on the left hand side of rules) on the sets from $\{X_i : i \in \Delta\}$ can be characterized by $\mu^{\mathbb{B}}_X(u)$, where $u \in INF_1(A(t), \Delta, \mathbb{A}^*)$, $X \in CLASS_{\mathbb{A}}(d)$, $\mathbb{B} = (U, A(t))$, and t is a prime implicant $f_{MG_1(\mathbb{A})}$.

Let us observe that the introduced certainty coefficients are computable from a given decision table. The problem of globally optimal decision rule generation is NP-hard (the proof is analogous as for the minimal reduct problem [10]) but it is possible to construct efficient heuristics generating decision rules in a form "near" to globally optimal.

In [12] β-lower and β-upper approximations of sets are defined for $0 \leq \beta < 0.5$ (the classical case [5] is obtained when $\beta = 0$). One can extend our method of optimal decision rules generation to this case. It is enough to consider instead of $\partial_{\mathbb{A}}$ a new decision attribute $\partial^{\beta}_{\mathbb{A}}$ and next to follow presented above procedures. The decision $\partial^{\beta}_{\mathbb{A}}$ is defined as follows:

if $\mu^{\mathbb{A}}_{X_i}(s) \geq 1 - \beta$ for some i
  then $\partial^{\beta}_{\mathbb{A}}(s) = \{i\}$
  else if $\{i : \beta < \mu^{\mathbb{A}}_{X_i}(s) < 1 - \beta\} = \emptyset$

then $\partial^{\beta}_{\mathbb{A}}(s) = \{0\}$
else $\partial^{\beta}_{\mathbb{A}}(s) = \{i : \beta < \mu^{\mathbb{A}}_{X_i}(s) < 1 - \beta\}$.

The discussed methods of decision rules generation are implemented in a system for classifying objects .

Conclusions

In the paper we have proposed a method for optimal decision rules generation with certainty coefficients. Our method can be applied also for generation of decision rules with minimal number of descriptors in each disjunct on the left hand side of rules.

At the end we would like to suggest some topics for further investigations.

Our procedures can in some cases generate a rule with many disjuncts on the left hand side of the decision rule. Moreover, each disjunct can be supported only by a few examples. In this case the attributes chosen for decision taking are inappropriate, namely they are not suitable for expressing the characteristic properties of the decision classes and a searching process for some new, more appropriate, attributes (classifiers) is necessary. In [11] is suggested a method searching for classifiers based on some multi-modal logics. The logic application for classifiers searching is an exciting research area.

We would like also to investigate logics with belief functions. The semantics of these logics is based on the, so called, decision table maps. They are Kripke models with worlds indexed by information vectors defined by a given decision table $\mathbb{A}$, the accessibility relation between worlds defined by the information inclusion relation and with special structures attached to any information vector. Any such structure is defined by a restriction of $\mathbb{A}$ to the information labelling that structure and contains restricted to that table belief functions. We investigate this kind of logics as candidates for expressing new classifiers, in particular we would like to verify a hypothesis that formulas of those logics are often suitable for expressing characteristic properties of object classes.

The well known rule of evidence combination from independent sources of information is the Dempster-Shafer rule [7]. In general, the combination rule should be based not only on the bpa functions but also on properties of knowledge embedded in both sources. An interesting problem arises to investigate an appropriate logic for this kind of reasoning.

References

1. R.K. Bhatnager, L.N. Kanal: Handling uncertain information: A review of numeric and non-numeric methods. In: L.N. Kanal, J.F. Lemmer (eds.): Uncertainty in Artificial Intelligence. Amsterdam: North - Holland 1986
2. F.M. Brown: Boolean reasoning. Dordrecht: Kluwer 1990
3. Y. Kodratoff, R. Michalski: Machine Learning: An Artificial Intelligence Approach, vol. 3. San Mateo: Morgan Kaufmann 1990

4. Z. Pawlak: Rough sets. International Journal of Information and Computer Science **11**, 344–356 (1982)
5. Z. Pawlak: Rough sets: Theoretical Aspects of Reasoning about Data. Dordrecht: Kluwer 1991
6. Z. Pawlak, A. Skowron: Rough membership functions. to appear In: M. Federizzi, J. Kacprzyk and R.R. Yager (eds.): Advances in the Dempster-Shafer Theory of Evidence. New York: John Wiley and Sons
7. G. Shafer: Mathematical Theory of Evidence. Princeton: University Press 1976
8. G. Shafer, J. Pearl: Readings in Uncertain Reasoning. San Mateo: Morgan Kaufmann 1990
9. A. Skowron, J.W. Grzymala-Busse: From rough set theory to evidence theory. to appear In: M. Federizzi, J. Kacprzyk and R.R. Yager (eds.): Advances in the Dempster-Shafer Theory of Evidence. New York: John Wiley and Sons
10. A. Skowron, C. Rauszer: The discernibility matrices and functions in information systems. In: R. Slowinski (ed.): Decision Support by Experience - Applications of the Rough Sets Theory. Dordrecht: Kluwer 1992, pp.331-362
11. A. Skowron, J. Stepaniuk: Searching for classifiers. In: M. De Glas, D. Gabbay: Proc. of First World Conference on Foundations of Artificial Intelligence, Paris, July 1-5, 1991. Paris: Angkor 1991, pp. 447–460
12. W. Ziarko: Analysis of uncertain information in the framework of variable precision rough set model. In: International Workshop Rough Sets: State of the Art and Perspectives, Poznan - Kiekrz (Poland), September 2-4, 1992. Extended Abstracts. Poznan 1992, pp.74–77

Upper and Lower Entropies of Belief Functions Using Compatible Probability Functions

C.W.R. Chau, P. Lingras and S.K.M. Wong

Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2

Abstract. This paper uses the compatible probability functions to define the notion of upper entropy and lower entropy of a belief function as a generalization of the Shannon entropy. The upper entropy measures the amount of information conveyed by the evidence currently available. The lower entropy measures the maximum possible amount of information that can be obtained if further evidence becomes available. This paper also analyzes the different characteristics of these entropies and the computational aspect. The study demonstrates usefulness of compatible probability functions to apply various notions from the probability theory to the theory of belief functions.

1 Introduction

The concepts of belief functions were originally derived from the upper and lower probabilities using a multivalued mapping [3] or a compatibility relation [14]. Compatibility relations can be used to develope a class of compatible probability functions for a given belief function [2, 5, 10]. The Bayesian theory of probability contains important concepts such as the Bayes rule of conditionalization and various information measures that are useful for making numeric judgments. This study illustrates how compatible probability functions can be used to apply similar concepts in the theory of belief functions.

The measurement of information contained in a belief structure is an important issue in any theory of partial belief [13]. Shannon [15] introduced the entropy function to measure the information contained in a Bayesian probability function. The Shannon entropy has been found useful in a variety of applications [6, 12]. In order to adopt Shafer's theory for similar applications, it is necessary to introduce an appropriate information measure for the belief functions. Shafer used the weight of evidence to estimate the information contained in a class of belief functions called the separable support functions. Unfortunately, Shafer's measure cannot be used for other categories of belief functions. Yager [19] proposed the entropy and specificity like measures which can be applied to all categories of belief functions. The entropy like measure provides an indication of the internal conflict in the given evidence, while specificity measures the non-specific uncertainty. These two measures are said to complement each other, but the relationship between them is not clearly described. It is also not very clear how one can use the measures suggested by Yager or its variations [4, 9, 16] to represent the amount of information contained in a belief function.

This study introduces the notion of *upper* and *lower* bounds of entropy of a belief function using compatible probability functions. The upper and lower entropies provide an objective criterion for measuring the information contained in a belief function. The upper entropy measures the information conveyed by the belief function based on existing evidence. The lower entropy, on the other hand, represents the maximum possible information that can be obtained by accumulation of evidence. Both the upper and lower entropies converge to the Shannon entropy for a Bayesian belief function. Thus, they provide a generalization of the concept of entropy for probability functions.

2 Brief Review

For completeness, we summarize here some of the basic notions in the theory of belief functions [13]. Some earlier work on information measures is also briefly reviewed.

2.1 Belief Functions

Based on an evidence, let $\Theta = \{\theta_1, \ldots, \theta_n\}$ denote the finite set of all possible answers to a question. We refer to Θ as the frame of discernment or simply the frame defined by this question. The power set of Θ, written 2^Θ, represents the set of all propositions discerned by Θ.

The belief functions introduced by Shafer [13] were originally derived from the concepts of upper and lower probabilities [3] which are useful for transferring the probability from one frame E to another frame Θ by using a multivalued mapping or a compatibility relation.

Definition 2.1:
Consider two frames of discernment E and Θ. An element $e_j \in E$ is *compatible* with an element $\theta_i \in \Theta$, written $e_j C \theta_i$, if the answer e_j to the question which defines E does not exclude the possibility that θ_i is an answer to the question which defines Θ. If an element $e_j \in E$ is *not* compatible with an element $\theta_i \in \Theta$, it is written as $e_j \not C \theta_i$.

Compatibility is symmetric: $e_j C \theta_i$ if and only if $\theta_i C e_j$. The compatibility relationship can be used to define the notion of *implication* between propositions from two different frames of discernment [11].

Definition 2.2:
A proposition $A \in 2^E$ is said to *imply* another proposition $B \in 2^\Theta$, written $A \Longrightarrow B$, if any element θ_i of Θ compatible with some $e_j \in A$ exists in B.

Definition 2.3:
A proposition $A \in 2^E$ is said to *exactly imply* another proposition $B \in 2^\Theta$, written $A \Longleftrightarrow B$, if A implies B but it does not imply any proper subset of B. Every $A \in 2^E$ exactly implies one and only one $B \in 2^\Theta$.

Consider a frame Θ which denotes the set of possible answers to a question. We are interested in obtaining a probability function $P : 2^{\Theta} \rightarrow [0,1]$ based on the given evidence. Suppose it is not possible to construct such a probability function directly, but from the given evidence we can define a frame - the *evidence frame* E. Moreover, let us assume that based on the evidence a probability function $P : 2^{E} \rightarrow [0,1]$ on frame E is known, and for simplicity let every $e_j \in E$ be compatible with at least one $\theta_i \in \Theta$. The issue here is how to use this knowledge about the evidence frame E to determine the degrees of belief in the propositions discerned by Θ.

In principle, the probability function P on frame Θ can be constructed from the probability function on frame E using the Bayes rule of conditionalization:

$$P(\{\theta_i\}) = \sum_{e_j \in E} P(\{e_j\}) \times P(\{\theta_i\}|\{e_j\}), \tag{1}$$

where $P(\{\theta_i\}|\{e_j\})$ are the conditional probabilities. Note that the expression $P(\{\ e_j\ \}) \times P(\{\ \theta_i\ \}|\{\ e_j\ \})$ is in fact equivalent to the joint probability $P(\{(\theta_i, e_j)\})$ defined on the frame $\Theta \times E$, i.e.,

$$P(\{(\theta_i, e_j)\}) = P(\{e_j\}) \times P(\{\theta_i\}|\{e_j\}). \tag{2}$$

In practice it is not always possible to provide an accurate estimation of the required conditional probabilities in the Bayes rule. In such situations, one may not be able to compute the probability $P(\{\theta_i\})$ from the Bayes rule as defined by equation 1. However, one may construct belief functions instead to estimate the degrees of belief in the propositions of 2^{Θ} as follows.

Given the probability function $P : 2^{E} \rightarrow [0,1]$ on the evidence frame E, we can define a function $m_E : 2^{\Theta} \rightarrow [0,1]$ for any proposition $F \in 2^{\Theta}$ as:

$$m_E(F) = \sum_{\{e_j\} \Leftrightarrow F} P(\{e_j\}). \tag{3}$$

The value $m_E(F)$ is the portion of the probability mass that is attributed to the union of those propositions in 2^{E} which *exactly imply* the proposition $F \in 2^{\Theta}$. A proposition F is called a focal element of the bpa if $m_E(F) > 0$. The value $m_E(F)$ measures the belief that one commits *exactly* to the proposition F. The total belief committed to a proposition A is given by:

$$Bel_E(A) = \sum_{F \subseteq A} m_E(F). \tag{4}$$

$Bel_E : 2^{\Theta} \rightarrow [0,1]$ is called a belief function. Another quantity referred to as the *plausibility*, written Pl, is defined by:

$$Pl(A) = 1 - Bel(\neg A), \tag{5}$$

where $\neg A$ denotes the negation of A.

In many instances the information about the evidence may be so vague that it is not even possible to explicitly construct the evidence frame. However, when

a basic probability assignment m_E defined on the propositions in 2^Θ is known, it is always possible to construct an *abstract* evidence frame E as follows. Let $F_1, \ldots, F_l$ be the focal elements of m_E. For every focal element F_j of m_E we assume that there is a unique $e_j \in E$ such that $\{e_j\} \Leftrightarrow F_j$, that is, $e_j C \theta_i$ for all $\theta_i \in F_j$ and $e_j \not C \theta_i$ for all $\theta_i \notin F_j$. This means that the number of elements in the abstract frame E will be the same as the number of focal elements of m_E. Thus the known bpa m_E can be viewed as a probability function defined on the abstract evidence frame E as:

$$P(\{e_j\}) = m_E(F_j). \tag{6}$$

Hereafter we will represent a belief function on a frame Θ in terms of an underlying probability function P defined on a distinct frame E and a compatibility relation C between frames E and Θ. It is understood that both the frame E and the compatibility relation C could be abstract constructs defined above, which may lack any semantics due to insufficient information.

One important feature in any theory of partial belief is the rule for combining evidence from different knowledge sources. The theory of belief functions uses the Dempster rule for combining two independent pieces of evidence [13, 11].

2.2 Information measures

Every probability or belief function conveys certain information about the frame of discernment Θ. A quantitative measure of this information can be very useful for comparing various belief or probability functions.

Shannon [15] proposed the entropy function $H : P \rightarrow [0, 1]$:

$$H(P) = -\sum_{\theta_i \in \Theta} P(\{\theta_i\}) \times log_n P(\{\theta_i\}) \tag{7}$$

to measure the amount of information conveyed by a given probability function P. The probability function $P : 2^\Theta \rightarrow [0, 1]$ conveys more information about the frame Θ if its entropy $H(P)$ is relatively small [15]. The entropy function H has been found very useful in various applications of the Bayesian probability theory [6, 12].

Shafer [13] introduced the weight of evidence to measure the information or evidence contained in a class of belief functions called *separable support functions*. The weight of evidence represents a gain in information with the accumulation of evidence. This is a desirable property for any reasonable measure of information. Unfortunately, the weight of evidence cannot be applied to a more general class of belief functions.

Yager [19] proposed two measures that can be applied to any category of belief functions. The entropy like measure E_m measures the internal conflict in an evidence. The specificity like measure S_m measures the non-specific uncertainty of a belief function. Several researchers [4, 9, 16] proposed variations of Yager's measures. However, it is not clear how one can use the pair (S_m, E_m) or its variations to measure the amount of information contained in a belief function.

3 Class of Compatible Probability Functions

The basic probability assignment $m(F_j)$ or the underlying probability function $P(\{e_j\})$ defined on the evidence frame E can be interpreted as the probability mass that can move freely within the subset F_j of Θ [13]. However, there is no knowledge about the exact allocation of the portions of $P(\{e_j\})$ to the individual elements $\theta_i \in F_j$. This type of non-specific uncertainty called ignorance is as inherent in the theory of belief functions as probabilistic uncertainty is in the Bayesian theory. Any additional knowledge that reflects the exact allocation of $p(\{e_j\})$ within F_j such as the conditional probability values $P(\{\theta_i\}|\{e_j\})$ will be helpful in reducing the ignorance. In fact, if such knowledge is known for every focal element of a belief function, then the degree of ignorance will decrease to zero and consequently the belief function is reduced to a Bayesian probability function as given by equation 1. In such a case, one can simply use the Shannon entropy function H defined by equation 7 to measure the degree of probabilistic uncertainty induced by the probability function P.

If some of the actual conditional probabilities $P(\{\theta_i\}|\{e_j\})$ are not available, the Shannon entropy function H cannot be applied. However, it may be possible to construct a class of probability functions $P(\{\theta_i\})$ on Θ using the probability function $P(\{e_j\})$ defined on the evidence frame E and the compatibility relation between E and Θ. The entropy function H can then be applied to this class of probability functions $P(\{\theta_i\})$.

Let $P(\{(\theta_i, e_j)\})$ given by equation 2 be the portion of the known probability mass $P(\{e_j\})$ allocated to θ_i in the focal element F_j. The constraints on $P(\{(\theta_i, e_j)\})$ are:

$$\sum_{\theta_i \in \Theta} P(\{(\theta_i, e_j)\}) = P(\{e_j\}), \text{ for every } e_j \in E, \tag{8}$$

$$P(\{(\theta_i, e_j)\}) = 0 \text{ if } \theta_i \not C\, e_j, \tag{9}$$

$$P(\{(\theta_i, e_j)\}) \geq 0 \text{ if } \theta_i C e_j. \tag{10}$$

Any probability function $P : 2^{E \times \Theta} \rightarrow [0,1]$ obeying these constraints is called an extension [7] of $P : 2^E \rightarrow [0,1]$ or the corresponding belief function Bel_E. In general, there will be an infinite number of such extensions for a belief function. Each one of these extensions uniquely determines a probability function $P : 2^\Theta \rightarrow [0,1]$, namely:

$$P(\{\theta_i\}) = \sum_{e_j \in E} P(\{(\theta_i, e_j)\}), \text{for every } \theta_i \in \Theta. \tag{11}$$

Any probability function $P : 2^\Theta \rightarrow [0,1]$ satisfying equation 11 is said to be *compatible* with the probability function $P : 2^E \rightarrow [0,1]$ or the corresponding belief function Bel_E. The compatible probability functions represent a class of Bayesian probability functions that can be obtained if additional information such as the conditional probabilities $P(\{\theta_i\}|\{e_j\})$ were available. In practice, these conditional probabilities are usually not available, especially if frame E is

an abstract construct as previously defined. The compatible probability functions were studied independently by Fagin and Halpern [5] and Lingras [10].

Original definition of belief and plausibility as upper and lower probability can be easily illustrated using class of compatible probability functions as follows.

$$Bel(A) = \min_P P(A), \text{for all } P : 2^\Theta \rightarrow [0,1] \text{compatible with } Bel. \quad (12)$$

$$Pl(A) = \max_P P(A), \text{for all } P : 2^\Theta \rightarrow [0,1] \text{compatible with } Bel. \quad (13)$$

The Bayesian theory of probability contains important concepts such as the Bayes rule of conditionalization and various information measures that are useful for making numeric judgments. This study illustrates how compatible probability functions can be used to apply similar concepts in the theory of belief functions.

4 Upper and lower entropies of belief function

Since a belief function can have an infinite number of compatible probability functions, it can therefore be associated with the entropy of any one of its compatible probability functions. We can characterize this range of entropy values by an upper and a lower bound, hereafter referred to as the upper and lower entropies of a belief function. Of course, we do not know the exact entropy value which a belief function would assume within this interval unless the conditional probabilities $P(\{\theta_i\}|\{e_j\})$ are known.

Definition 4.1
The upper entropy of a belief function Bel, written $\overline{H}(Bel)$, is given by:

$$\overline{H}(Bel) = \max_P H(P), \text{for all } P : 2^\Theta \rightarrow [0,1] \text{compatible with } Bel. \quad (14)$$

The lower entropy of a belief function Bel, written $\underline{H}(Bel)$, is given by:

$$\underline{H}(Bel) = \min_P H(P), \text{for all } P : 2^\Theta \rightarrow [0,1] \text{compatible with } Bel. \quad (15)$$

As mentioned before, the amount of information contained in a probability function is inversely proportional to its entropy value. The upper entropy, which corresponds to the most unbiasedly distributed probability function compatible with a belief function, provides a measure of the amount of information contained in the belief function. High values of the upper entropy indicate the fact that the amount of information contained in the belief function is relatively low. Since $0 \leq H(P) \leq 1$, the belief function with an upper entropy $\overline{H}(Bel) = 1$ contains the minimum amount of information. On the other hand, the lower entropy, which corresponds to the most biased probability function compatible with a belief function, represents the maximum possible amount of information that can be obtained if future evidence becomes available. Low values of the lower entropy indicate the fact that the maximum possible amount of information that can be obtained from the belief function by accumulating future evidence

is relatively high. Theoretically, a belief function with a lower entropy of zero can potentially lead to a unique outcome.

The upper and lower entropies of a belief function show monotonic changes as evidence is being accumulated. In the following subsection we will study the behavior of the upper and lower entropies with the accumulation of consistent evidence.

4.1 Accumulation of consistent evidence

Consider belief functions Bel_E and Bel_S defined by two different pieces of evidence. Let $Bel_E \oplus Bel_S$ be the result of the combination of Bel_E and Bel_S. The accumulation of consistent evidence should in general lead to an increase in the amount of information about the discerned frame. The weight of evidence suggested by Shafer, by construction, reflects such an increase in information. The following theorem shows that the upper and lower entropies indeed correctly reflect the accumulation of consistent evidence.

Theorem 4.1
Let Bel_E and Bel_S be the belief functions representing two pieces of evidence *consistent* [10] with each other. Let $Bel_E \oplus Bel_S$ be the belief function representing the combined evidence. Then,

$$\overline{H}(Bel_E \oplus Bel_S) \leq \min(\overline{H}(Bel_E), \overline{H}(Bel_S)), and \tag{16}$$

$$\underline{H}(Bel_E \oplus Bel_S) \geq \max(\underline{H}(Bel_E), \underline{H}(Bel_S)). \tag{17}$$

Theorem 4.1 is proved in [2].

According to Theorem 4.1, the upper entropy decreases with the combination process. This behavior reflects a gain in information by pooling consistent evidence from different sources. The lower entropy, on the other hand, provides a measure of the maximum possible amount of information that can be obtained by the accumulation of consistent evidence. The range from $\overline{H}(Bel)$ to $\underline{H}(Bel)$ provides a measure of the possible gain in information that can be achieved by accumulating evidence. Thus, the difference $\Delta H = \overline{H}(Bel) - \underline{H}(Bel)$ can be considered as a measure of ignorance. As expected, ignorance is reduced with the accumulation of evidence.

The above discussion provides a plausible justification for the use of upper and lower entropies and the difference ΔH as measures of information and ignorance. In the following section we will illustrate how these measures can be used to characterize different categories of belief functions.

4.2 Different categories of belief functions

Information conveyed by different categories of belief functions [13] has different characteristics. The upper and lower entropies enable us to identify these characteristics conveniently.

Theorem 4.2
Let Bel be a belief function with $\underline{H}(Bel)$ and $\overline{H}(Bel)$ as its lower and upper entropies. Then

1. If Bel is a consonant support function, then $\underline{H}(Bel) = 0$.
2. If Bel is a vacuous belief function, then $\underline{H}(Bel) = 0$ and $\overline{H}(Bel) = 1$.
3. If Bel is a Bayesian belief function, then $\underline{H}(Bel) = \overline{H}(Bel) = H(Bel)$, where

$$H(Bel) = -\sum_{\theta_i \in \Theta} Bel(\{\theta_i\}) \log_n Bel(\{\theta_i\}) = -\sum_{\theta_i \in \Theta} m(\{\theta_i\}) \log_n m(\{\theta_i\}) \tag{18}$$

Proof of the theorem can be found in [2].

Based on Theorem 4.2, one can say that the lower and upper entropies provide an intuitively appealing description of the amount of information contained in a belief function. For example, a *consonant support function* represents an evidence pointing in a single direction [13]. Hence, the combination of a consonant support function with other belief functions which are also pointing in the same direction should eventually lead to a unique outcome, i.e. a belief function with $Bel(\{\theta_i\}) = 1$ for some $\theta_i \in \Theta$. The fact that the lower entropy is equal to 0 for consonant support functions correctly reflects this characteristic. More importantly, the upper entropy enables us to compare the information contained in different consonant support functions. The *vacuous belief function* is a special type of consonant support function [13]. Hence, the above discussion is also applicable to the vacuous belief function. The upper entropy of the vacuous belief function is equal to 1, which means that the vacuous belief function conveys no information. Also, the difference between the upper entropy and the lower entropy (ΔH) for the vacuous belief function is equal to 1. This observation appropriately reflects the fact that the vacuous belief function represents maximum ignorance. A Bayesian belief function, on the other hand, represents an evidence with minimum ignorance (which is equal to 0) because every focal element is a singleton. This is consistent with the observation that the upper entropy of a Bayesian belief function is equal to its lower entropy (i.e. $\Delta H = 0$).

4.3 Computation of the upper and lower entropies

There are many numerical methods we can use to compute the joint probability function by maximizing the entropy function defined by equation 7 [1, 8]. On the other hand, the entropy function cannot be minimized using standard numerical methods. In order to determine the lower entropy, we have to find the most biased compatible probability function. An algorithm for determining the lower entropy defined by the most biased probability function that is compatible with a belief function is proposed in [2]. There may be many solutions for $P(\{(\theta_i, e_j)\})$ which maximize/minimize the entropy $H(P)$. However, we are only interested in the maximum/minimum value of $H(P)$, which is unique. The

time complexity of the above algorithms for the worst case can be very high. However, it may be possible to increase the efficiency of these algorithms using non-linear programming techniques.

5 Conclusion

Compatible probability functions can be used to apply some of the important notions from the probability theory to the theory of belief functions. This paper introduces the notion of upper and lower entropies of compatible probability functions to measure the amount of information conveyed by a belief function. The upper entropy is the entropy of the most unbiased probability function compatible to the belief function. It denotes the amount of information conveyed by the evidence currently available. The lower entropy, on the other hand, is the entropy of the most biased probability function compatible to the belief function. It denotes the maximum possible amount of information that can be obtained if further evidence becomes available. The upper and lower entropies provide an objective criterion for analyzing the information conveyed by a piece of evidence. The difference between the upper and the lower entropies indicates the ignorance induced by the evidence. The upper entropy, lower entropy and the difference between them accurately reflects the information gain from the accumulation of consistent evidence.

References

1. D.P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, Inc., New York, pp. 71-75, 1982.
2. C.W.R. Chau, P.J. Lingras and S.K.M Wong, Upper and Lower Entroipes of Belief Functions, *Technical Report*, Department of Computer Science, University of Regina, Sask., Canada, 1990.
3. A. Dempster, Upper and Lower Probabilities Induced by a Multivalued Mapping, *Annals of Mathematical Statistics*, 38, pp. 325-339, 1967.
4. D. Dubois and H. Prade, Properties of measures of information in evidence and possibility theories, *Fuzzy Sets and Systems*, 24, 1987.
5. Fagin, R. and Halpern, J. (1990). A New Approach to Updating Beliefs, *Proceeding of the Sixth Conference on Uncertainty in AI*, Cambridge, Mass., July 27-29, pp. 317-325.
6. S. Guiasu, *Information Theory with Applications*, McGraw-Hill, London, 1977.
7. J. Hartmanis, The Application of Some Basic Inequalities for Entropy, *Information and Control*, Vol. 2, pp. 199-213, 1959.
8. IMSL, Nonlinearly Constrained Minimization Using finite-different gradient, *IMSL Math/Library User's Manual*, pp. 895-908, 1987.
9. G.J. Klir and T.A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.
10. P.J. Lingras, *Qualitative and Quantitative Reasoning Under Uncertainty in Intelligent Information Systems*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Regina, Sask., Canada, 1991.

11. P.J. Lingras and S.K.M. Wong, Two Perspectives of the Dempster-Shafer Theory of Belief Functions, to appear in *The International Journal of Man-Machine Studies*, 1990.
12. J.R. Quinlan, Inductive inference as a tool for the construction of high-performance programs, *Machine Learning*, R.S. Michalski, T.M. Mitchell and J. Carbonell eds. Palo Alto, CA: Tioga, 1983.
13. G. Shafer, *A Mathematical Theory of Evidence*, Princeton. N.J.: Princeton University Press, 1976.
14. G. Shafer, Belief functions and possibilities measures, *The Analysis of Fuzzy Information 1*, Bezdek, J.C., CRC Press, 1986.
15. C.E. Shannon, A mathematic theory of communication, *Bell Technical Journal*, Vol. 4, pp. 379-423, 1948.
16. H.E. Stephanou and S. Lu, Measuring Consensus Effectiveness by a Generalized Entropy Criterion, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 4, pp. 544-554, July 1988.
17. S.K.M. Wong and P.J. Lingras, Unification of the Bayes Conditionalization and the Dempster Rule by Minimizing Information Gain, to appear in *The proceedings of the Sixth Conference on Uncertainty in AI*, Cambridge, Mass., July 27-29 1990.
18. S.K.M. Wong, W. Ziarko and R. Le Ye, Comparison of rough-set and statistical method in inductive learning, *International Journal of Man-Machine Studies*, 25, pp. 53-72, 1986.
19. R.R. Yager, Entropy and Specificity in a Mathematical Theory of Evidence, *International Journal of General Systems*, Vol. 9, pp. 249-260, 1983.

Reasoning about Higher Order Uncertainty in Possibilistic Logic

Churn Jung Liau[1] and Bertrand I-Peng Lin[2]

[1] Institute of Information Science, Academia Sinica,
Taipei, Taiwan, ROC
[2] Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, ROC

Abstract. Possibilistic logic is an important approach for reasoning about possibility and necessity. By formulating possibilistic reasoning as a kind of modal logic, we can represent and reason about higher order uncertainty to any nested degrees. In this paper, we present a system with built-in axioms for reasoning about higher order uncertainty. Some intuition behind the system is discussed and the soundness and completness of it with respect to the class of finite transitive and serial (in fuzzy set-theoretic sense) possible-world models are proved.

1 Introduction

Possibilistic logic[1] is an important approach for reasoning about possibility and necessity. In this logic, two certainty measures, called possibility and necessity measures respectively, are assigned to the well-formed formulas of classical logic. Thus, if f is a well-formed formula (wff) of propositional logic, then $(f(Nc))$ and $(f(\Pi c))$ $(c \in [0,1])$ are wffs of possibilistic logic. In this way, the uncertainty of a crisp proposition can easily be represented. However, it is usually difficult to estimate the possibility (and dually, the necessity) of a proposition precisely. Thus, the possibility and necessity of a proposition are themselves subject to judgement and evaluation. For example, given the information "John is tall", we might estimate that the necessity of John's height being over six feet is at least 0.8. However, we may be highly suspect the correctness of the estimation, so we will talk about the uncertainty of the estimation. This kind of uncertainty about the estimation is called higher order uncertainty. By formulating possibilistic logic as a kind of multi-modal language and exploiting the notions of nested modal operators, we can represent higher order uncertainty naturally. In [4], we have developed such a logic and its axiomatic system, called quantitative modal logic (QML). The remaining problem is what additional axioms are needed for reasoning about higher order uncertainty. The paper will be devoted to the solution of this problem.

In the next section, we will first review the underlying logic—QML. Then, the axioms for higher order uncertainty reasoning are presented and their implications are also discussed. Finally, the completeness of the extended axiomatic system is considered.

2 Review of Quantitative Modal Logic

2.1 Syntax

QML is an extension of propositional logic with four classes of quantitative modal operators: $\langle c\rangle$, $\langle c\rangle^+$, $[c]$ and $[c]^+$ for all $c \in [0,1]$.

The syntactic formation rules of QML consist of those for propositional logic and the following one.

- if f is a wff, so are $\langle c\rangle f$, $\langle c\rangle^+ f$, $[c]f$ and $[c]^+ f$ for all $c \in [0,1]$.

We usually use lower case letters (sometimes with indices) p, q, r to denote atomic formulas and f, g, h to denote wffs. We also assume all classical logical connectives are available, either as primitives or as abbreviations.

2.2 Semantics

The semantics of QML is a generalization of the standard Kripke semantics for modal logic[3]. Define a *possibility frame* $F = \langle W, R\rangle$, where W is a set of possible words and $R : W^2 \to [0,1]$ is a fuzzy accessibility relation on W. Let PV and FA denote the set of all propositional variables and the set of all wffs respectively. Then a model of QML is a triple $M = \langle W, R, TA\rangle$, where $\langle W, R\rangle$ is a possibility frame and $TA : W \times PV \to \{0,1\}$ is a truth value assignment for all worlds. A proposition p is said to be true at a world w iff $TA(w,p) = 1$. Given R, we can define a possibility distribution R_w for each $w \in W$ such that $R_w(s) = R(w,s)$ for all s in W. Similarly, we can also define TA_w for each $w \in W$ such that $TA_w(p) = TA(w,p)$ for all p in PV. Thus, mathematically, a model can be equivalently written as $\langle W, \langle R_w, TA_w\rangle_{w\in W}\rangle$. Given a model $M = \langle W, R, TA\rangle$, we can define the satisfication relation $\models_M \subseteq W \times FA$ as follows. First, let us define $N_w(f) = \inf\{1 - R(w,s) \mid s \models_M \neg f, s \in W\}$. Consequently, N_w is just the necessity measure induced by the possibility distribution R_w defined by Dubois and Prade[1]. Then, $w \models_M [c]f$ (resp. $w \models_M [c]^+ f$) iff $N_w(f) \geq c$ (resp. $> c$). The satisfication relations for $\langle c\rangle f$ and $\langle c\rangle^+ f$ are defined analogously by replacing $N_w(f)$ with $\Pi_w(f) = 1 - N_w(\neg f)$. Here, for convenience, we define $\sup\emptyset = 0$ and $\inf\emptyset = 1$. Furthermore, the satisfication of all other wffs are defined as usual in classical logic.

A wff f is said to be valid in $M = \langle W, R, TA\rangle$, writen $\models_M f$ iff for all $w \in W, w \models_M f$. If S is a set of wffs, then $\models_M S$ means that for all $f \in S$, $\models_M f$. If $\mathbf{C}$ is a class of models and S is a set of of wffs, then we write $S \models_{\mathbf{C}} f$ to mean that for all $M \in \mathbf{C}$, $\models_M S$ implies $\models_M f$. A model is called serial if $\forall w \in W, \sup_{s\in W} R(w,s) = 1$ and finite if W is finite.

2.3 Axiomatic system for possibilistic reasoning

An axiomatic system for the class of all serial models have been developed in [4] and we call it D. The system consists of the following axiomatic schemata and rules of inference.

1. Inequality constraints:
   (a) AX1 (monotonicity):

$$[c]f \supset [d]^+f, \text{ if } c > d$$

   (b) AX2 (dichotomy):

$$[c]^+f \supset [c]f$$

   (c) AX3 (lower and upper bounds):

$$[0]f, \neg[1]^+f$$

2. Propositional reasoning:
   AX4: a set of complete axioms for propositional logic.
3. Possibility and necessity (AXK):
   (a) $[c](f \supset g) \supset ([c]f \supset [c]g)$.
   (b) $[c]^+(f \supset g) \supset ([c]^+f \supset [c]^+g)$.
4. Seriality (AXD):

$$[0]^+f \supset \langle 1\rangle f$$

5. Rules of inference:
   (a) R1 (Modus Ponens):

$$\frac{f, f \supset g}{g}$$

   (b) R2 (Rule of necessitation):

$$\frac{f}{[1]f}$$

Formally, if **D** is the class of all serial models, then for any finite set B of wffs and wff f, we have $B \models_{\mathbf{D}} f$ iff f is derivable from B in the system D.

3 Axioms for Higher Order Uncertainty Reasoning

In [4], it is also shown that the system D is suitable for possibilistic reasoning. However, to handle higher order uncertainty, additional axioms may be needed. The question is " what are the axioms just needed ?". There may be many answers to this question depending on the meaning of necessity and possibility. In traditional modal logic, the most basic axiom about nested modality is as follows:

$$\Box f \supset \Box\Box f.$$

This axiom has an intuitive interpretation in epistemic logic. It says that if some agent knows f, then he knows that he knows f. The axiom is valid in all transitive possible world models for ordinary modal logic. Thus, it is natural to investigate the corresponding model class in the QML case. Define a model for QML as (sup-min) transitive iff for all $w, w' \in W$, we have $R(w, w') \geq \sup_{s \in W} \min(R(w, s), R(s, w'))$.

Parallel to the above-mentioned axiom for nested modality, we can prove that the following axioms are valid in all transitive models:

$$AX4^{w}.1 : [c]f \supset [c][c]f$$

and

$$AX4^{u}.2 : [c]^{+}f \supset [c][c]^{+}f.$$

However, the axiom $AX4^{w}.2$ is somewhat too weak to obtain a complete axiom system for transitive models. The main reason is that we consider finite as well as infinite models, while our inference rules and axioms are essentially finitary. Thus, if we concern only the finite models, the axiom can be strenghtened as follows:

$$AX4^{s}.2 : [c]^{+}f \supset [c]^{+}[c]^{+}f.$$

We define $D4$ as the union of D and the axioms $AX4^{u}.1$ and $AX4^{s}.2$. Then $D4$ has the following reasonable derived axioms:

$$\langle c\rangle\langle d\rangle f \supset \langle \min(c,d)\rangle f,$$

$$\langle c\rangle^{+}\langle d\rangle^{+} f \supset \langle \min(c,d)\rangle^{+} f.$$

Consequently, the axioms will allow us to collapse sequence of possibility operators into a single one.

4 Soundness and Completeness

4.1 Soundness

We will prove that the system $D4$ is sound and complete for the class of all finite serial transitive models in this section.

Let us consider the soundness theorem firstly. It is based on the following two lemmas.

Lemma 1. *If* $\mathbf{C}$ *is a class of transitive models, then*

(1) $\models_{\mathbf{C}} [c]f \supset [c][c]f$
(2) $\models_{\mathbf{C}} [c]^{+}f \supset [c][c]^{+}f$

Proof:

(1) Let $M \in \mathbf{C}$ and $w \models_M [c]f$. Define

$$W_1 = \{u \mid u \models_M \neg[c]f\}$$

$$W_2 = \{u \mid u \models_M \neg f\}.$$

Then, by assumption and definition,

$$\inf_{u \in W_2} (1 - R(w,u)) \geq c$$

and

$$\forall w' \in W_1, \inf_{u \in W_2} (1 - R(w', u)) < c.$$

This implies $\sup_{u \in W_2} R(w, u) \leq 1 - c$, and $\sup_{u \in W_2} R(w', u) > 1 - c, \forall w' \in W_1$. Then, by transitivity of R

$$R(w, u) \geq \min(R(w, w'), R(w', u)), \forall w', u \in W.$$

Thus,

$$\begin{aligned} \sup_{u \in W_2} R(w, u) &\geq \sup_{u \in W_2} \min(R(w, w'), R(w', u)) \\ &= \min(R(w, w'), \sup_{u \in W_2} R(w', u)) \end{aligned}$$

Now,

$$\min(R(w, w'), \sup_{u \in W_2} R(w', u)) \leq \sup_{u \in W_2} R(w, u) \leq 1 - c,$$

so $R(w, w') \leq 1 - c, \forall w' \in W_1$, since $\sup_{u \in W_2} R(w', u) > 1 - c, \forall w' \in W_1$. This results in $\inf_{w' \in W_1}(1 - R(w, w')) \geq c$, i. e. $w \models_M [c][c]f$.

(2) Assume $w \models_M [c]^+ f$, we redefine $W_1 = \{u \mid u \models_M \neg[c]^+ f\}$. Then, following an analogous argument as above, we have

$$R(w, w') < 1 - c, \forall w' \in W_1,$$

and this results in $\inf_{w' \in W_1}(1 - R(w, w')) \geq c$. That is, $w \models_M [c][c]^+ f$

□

Lemma 2. *If* **C** *is a class of finite transitive models, then* $\models_{\mathbf{C}} [c]^+ f \supset [c]^+[c]^+ f$.

Proof: In the proof of Lemma 1(2), we can obtain $\inf_{w' \in W_1}(1 - R(w, w')) > c$ at the last step, since when W is finite, the function "inf" can be replaced by 'min".□

Let **D4** denote the class of all finite serial and transitive models for QML, then we have the following theorem.

Theorem 3 (Soundness). *Let S be a set of wffs and f be a wff. If f is derivable from S in system D4, then* $S \models_{\mathbf{D4}} f$.

Proof: The preceding lemmas shows that axioms $AX4^w.1$ and $AX4^s.2$ are valid in **D4**. Combining this with the soundness theorem of the system D prove the required result.

4.2 Completeness

We adopt a Henkin-style construction to prove the completeness. This is esentially analogous to the method used by Fitting[2] in proving the completeness for ordinary modal logic. The critical difference is induced by the fuzziness of accessibility relation.

To simplify the preesentation, we use the uniform notations for modal logic. The uniform notations are based on a classification of the non-literal wffs into six categories according to the formula's main connective which is used to combine its immediate subformulas. We list the classification in Table 1-4. We will abuse the notation and use $\alpha, \alpha_1, \cdots$, etc, to deonte wffs of the respective types and their immediate subformulae

Table 1. α wffs and their component formulas

α	α_1	α_2
$f \wedge g$	f	g
$\neg(f \vee g)$	$\neg f$	$\neg g$
$\neg(f \supset g)$	f	$\neg g$
$\neg\neg f$	f	f

Table 2. β wffs and their component formalas

β	β_1	β_2
$f \vee g$	f	g
$\neg(f \wedge g)$	$\neg f$	$\neg g$
$f \supset g$	$\neg f$	g

Table 3. ν and ν^+ wffs (with parameter c) and their component formulas

$\nu(c)$	$\nu_0(c)$
$[c]f$	f
$\neg\langle 1-c\rangle^+ f$	$\neg f$

$\nu^+(c)$	$\nu_0^+(c)$
$[c]^+ f$	f
$\neg\langle 1-c\rangle f$	$\neg f$

The completeness theorem is based on a more general theorem, called model existence theorem. To state and prove the model existence theorem, we need some further definitions.

Table 4. π and π^+ wffs (with parameter c) and their component formulas

$\pi(c)$	$\pi_0(c)$	$\pi^+(c)$	$\pi_0^+(c)$
$\langle c\rangle f$	f	$\langle c\rangle^+ f$	f
$\neg[1-c]^+ f$	$\neg f$	$\neg[1-c]f$	$\neg f$

Definition 4. Let S be a set of wffs. Then

1. Define the positive subformulas of S, denoted by S^+, as the smallest set containing S and being closed under the following conditions:
   (a) if $\neg f, \langle c\rangle f, \langle c\rangle^+ f, [c]f$, or $[c]^+ f \in S^+$, then $f \in S^+$.
   (b) if $f \wedge g, f \vee g, f \supset g \in S^+$, then $f, g \in S^+$.
2. Define the negation subformulas of S, denoted by S^-, as the following set:

$$S^- = \{\neg f \mid f \in S^+\}.$$

3. Define $Sub(S) = S^+ \cup S^-$.
4. Define the parameter value set of S, $V(S)$ as follows:

$$V(S) = \{c, 1-c \mid \exists f \text{ such that } [c]f, \langle c\rangle f, [c]^+ f, or \langle c\rangle^+ f \in Sub(S)\} \cup \{0, 1\}.$$

Also, $Sub(f) = Sub(\{f\})$ and $V(f) = V(\{f\})$ if f is a single wff. Note that if S is finite, then $Sub(S)$ and $V(S)$ are, too. In what follows, let $F = Sub(G)$ for some finite set G.

Definition 5. Let $\Theta \subseteq 2^F$ be a collection of subsets of F. Then Θ is called a classical consistency property on F iff for all $S \in \Theta$, S satisfies the following three conditions:

(a) $\alpha \in S \Rightarrow S \cup \{\alpha_1, \alpha_2\} \in \Theta$,
(b) $\beta \in S \Rightarrow S \cup \{\beta_1\} \in \Theta$, or $S \cup \{\beta_2\} \in \Theta$, and
(c) for any atomic wff p, S does not contain p and $\neg p$ simultaneously, and none of the following wffs are in S : $\nu^+(1), \pi^+(1), \neg\top, \bot$.

Definition 6. Define the world-alternative function H and strict world-alternative function H^+ on F as follows.
$H, H^+ : 2^F \times V(F) \to 2^F$,

$$H(S, c) = \{\nu_0(c'), \nu(c') \mid c' \geq c, \nu(c') \in S\} \cup \{\nu_0^+(c'), \nu^+(c') \mid c' \geq c, \nu^+(c') \in S\}$$

and

$$H^+(S, c) = \{\nu_0(c'), \nu(c') \mid c' > c, \nu(c') \in S\} \cup \{\nu_0^+(c'), \nu^+(c') \mid c' \geq c, \nu^+(c') \in S\}.$$

Definition 7. Let $\Theta \subseteq 2^F$ be a collection of subsets of F. Then Θ is a *D4-consistency property* iff it is a classical consistency property and for all $S \in \Theta$, S satisfies the following three additional conditions:

(h1) $c > 0$ and $\pi(c) \in S \Rightarrow H^+(S, 1-c) \cup \{\pi_0(c)\} \in \Theta$, and

(h2) $\pi^+(c) \in S \Rightarrow H(S, 1-c) \cup \{\pi_0^+(c)\} \in \Theta$.
(d4) $H^+(S, 0) \in \Theta$.

Let $B \subseteq F$, then a $D4$-consistency property Θ is called *B-compatible* iff for all $S \in \Theta$, and $f \in B$, $S \cup \{f\} \in \Theta$.

Depending on these definitions, we can state and prove the main theorem.

Theorem 8 (Model Existence theorem). *If F is a finite set of wffs, $B \subseteq F$, and Θ is a B-compatible D4-consistency property on F, then there exists a finite serial and transitive model $M = \langle W, R, TA \rangle$ such that $\models_M B$ and for all $f \in S \in \Theta$, there exists a world $w \in W$ such that $w \models_M f$.*

proof: To prove this theorem, we use a constructive argument. First, noting that Θ is ordered by $\subseteq$ and all elements of Θ is finite sets, we can construct the model $M_\Theta = \langle W, R, TA \rangle$ as follows:

W: the set of maximal members in Θ,
$TA(S, p) = 1 \Leftrightarrow p \in S, \forall S \in W$ and $p \in PV$, and
R must satisfy that for all $S, S' \in W$, and $c \in V(F)$:
(r1) if $c \neq 0$, then $H(S, c) \subseteq S' \Leftrightarrow R(S, S') > 1 - c$
(r2) $H^+(S, c) \subseteq S' \Leftrightarrow R(S, S') \geq 1 - c$.

Then, we use a series of lemmas to show that M_Θ is well-defined and satisfies the requirement of the theorem.

Lemma 9. *There exists a transitive R satisfying (r1) and (r2).*

proof: According to Definition 6, we have the following results for all $S \in 2^F$ and $c, c' \in V(F)$:

(i) $H^+(S, c) \subseteq H(S, c)$,
(ii) if $c \geq c'$, then $H(S, c) \subseteq H(s, c')$,
(iii) if $c \geq c'$, then $H^+(S, c) \subseteq H^+(S, c')$, and
(iv) if $c > c'$, then $H(S, c) \subseteq H^+(S, c')$
(v) for all $S_1, S_2, S_3 \in 2^F$, if $H^+(S_1, c) \subseteq S_2$ and $H^+(S_2, c) \subseteq S_3$, then $H^+(S_1, c) \subseteq H^+(H^+(S_1, c), c) \subseteq H^+(S_2, c) \subseteq S_3$.
(vi) for all $S_1, S_2, S_3 \in 2^F$, if $H(S_1, c) \subseteq S_2$ and $H(S_2, c) \subseteq S_3$, then $H(S_1, c) \subseteq S_3$.

Let $V(F) = \{c_1, c_2, c_3, \cdots, c_n, \}$ such that $1 = c_1 > c_2 > c_3 > \cdots > c_n = 0$. Then, we have the following inclusion chain for any $S \in 2^F$:

$$\emptyset = H^+(S, c_1) \subseteq H(S, c_1) \subseteq \cdots \subseteq H^+(S, c_i) \subseteq H(S, c_i) \subseteq H^+(S, c_{i+1}) \subseteq \cdots \subseteq H^+(S$$

Thus, given S and S', there are three possible cases:

Case 1: for some i such that $1 \leq i \leq n-1$, $H^+(S, c_i) \subseteq S'$ and $H(S, c_i) \not\subseteq S'$. In this case, the only value of $R(S, S')$ which satisfies the conditions (r1) and (r2) is $1 - c_i$.

Case 2: for some i such that $1 \leq i \leq n-1$, $H(S, c_i) \subseteq S'$ and $H^+(S, c_{i+1}) \not\subseteq S'$. In this case, if $1 - c_i < R(S, S') < 1 - c_{i+1}$ then (r1) and (r2) can be satisfied. Thus, we can set $R(S, S') = 1 - \frac{c_i + c_{i+1}}{2}$

Case 3: if $H^+(S, c_n) \subseteq S'$, then $R(S, S') = 1$

Then, according to (v) and (vi) above, we can see that for all $S_1, S_2, S_3 \in W$, we have $R(S_1, S_3) \geq \min(R(S_1, S_2), R(S_2, S_3))$, i.e., R is a transitive fuzzy relation. □

Lemma 10. *M_Θ is a serial model.*

proof: Since for any S in W, $H^+(S, 0) \in \Theta$ by condition (d), and each element of Θ is a finite set, we can find the maximal extension of $H^+(S, 0)$, say S', in Θ. Then $S' \in W$, and $H^+(S, 0) \subseteq S'$, so $R(S, S') = 1$ and $\sup_{S' \in W} R(S, S') = 1$. □

Lemma 11. *For any $S \in W$ and wff $f \in S$, $S \models_{M_\Theta} f$.*

proof: By induction on the structure of f. For convenience, we drop the subscript M_Θ in the following proof, and write $S \models f$ only. We consider the following exhaustive cases.

Case 1: f is an atomic wff. By the definition of M_Θ, $TA(S, f) = 1$, since $f \in S$. Thus, $S \models f$ according to the possible world semantics.

Case 2: $f = \neg p$ for some atomic wff p. By condition (c) of Definition 5, $\neg p \in S$ implies $p \notin S$, and so $TA(S, p) = 0$. This in turn implies $S \models f$.

Case 3: f is an α wff. Then $S \cup \{\alpha_1, \alpha_2\} \in \Theta$, and since S is a maximal member of Θ, this means $\alpha_1, \alpha_2 \in S$. By induction hypothesis, $S \models \alpha_1$ and $S \models \alpha_2$, so $S \models f$.

Case 4: f is a β wff. Then $S \cup \{\beta_1\} \in \Theta$ or $S \cup \{\beta_2\} \in \Theta$, so $S \models \beta_1$ or $S \models \beta_2$ by the maximality of S. Thus, $S \models f$.

Case 5:f is a $\pi(c)$ wff with $c > 0$. (note that if $c = 0$, $S \models \pi(c)$ naturally). Then $H^+(S, 1 - c) \cup \{\pi_0(c)\} \in \Theta$, so let S' be the maximal extension of $H^+(S, 1 - c) \cup \{\pi_0(c)\}$ in Θ, we have $R(S, S') \geq c$ since $H^+(S, 1 - c) \subseteq S'$. Furthermore, $\pi_0(c) \in S'$, so $S' \models \pi_0(c)$ by induction hypothesis. Thus , $S \models f$, by the definition of $\pi_0(c)$ wffs and their semantics.

Case 6:f is a $\pi^+(c)$ wff with $c < 1$. (Note that if $c = 1$, then $\pi^+(1) \notin S$). This case is similar to case 5.

Case 7: f is a $\nu(c)$ wff. Let S' be a element of W. If $S' \models \neg\nu_0(c)$, then $\nu_0(c) \notin S'$ by induction hypothesis. This implies $H(S\ c) \not\subseteq S'$ since $\nu_0(c) \in H(S, c)$ by definition, and in turn $1 - R(S, S') \geq c$ by the definition of M_Θ. Thus we have $\inf\{1 - R(S, S') \mid S' \models \neg\nu_0(c), S' \in W\} \geq c$, i. e. $S \models f$.

Case 8:f is a $\nu^+(c)$ wff. The proof is similar to that of Case 7, but use the property of W being finite in the last infernce step.□

Note that our induction basis are literals (i. e. Case 1 and 2), not just atomic formulas, so we can infer the results in all other cases.

Now, we can complete the proof of the model existence theorem. First, since Θ is B-compatible, we have $B \subseteq S$ for any $S \in W$, by the maximality of S. Thus for all $S \in W, S \models f$ if $f \in B$ by Lemma 11. That is, $\models_{M_\Theta} B$. Moveover, since for all $f \in S \in \Theta$, there exists a maximal extension of S, say S', in W, so $S' \models f$ by Lemma 11. Finally, W is finite and M_Θ is transitive and serial by Lemma 9 and 10. □

We can now consider the completeness of the $D4$ system.

Theorem 12. *Let B be a finite set of wffs and f be a wff. Then $B \models_{\mathbf{D4}} f$ implies that f is derivable from B in $D4$.*

Proof: Let $F = Sub(B \cup \{f\})$. Define $\Theta = \{S \subseteq F \mid B \not\vdash_{D4} \neg \bigwedge S\}$, where "$\bigwedge S$" denote the conjunction of all wffs in S and $B \not\vdash_{D4} f$ means that f is not derivable from B in $D4$. It can be shown that Θ is a B-compatible $D4$-consistency property on F. Thus if $B \not\vdash_L f$, then $\{\neg f\} \in \Theta$, and by the model existence theorem, we have $B \not\models_{\mathbf{D4}} f$. □

5 Concluding Remarks

We have provided a system with built-in axioms for higher order uncertainty reasoning. However, we do not claim that $D4$ is the unique possibile system for this purpose. In fact, if more constraints are imposed on the possible models, then some additional axioms can be derived. For example, if we require that the fuzzy accessibility relation is a similarity relation as in [5], then two additional axioms, $[0]^+ f \supset f$ (corresponding to reflexivity) and $f \supset [1-c]\langle c\rangle^+ f$ (corresponding to symmetry) could be added to $D4$.

References

1. D. Dubois and H. Prade. "An introduction to possibilistic and fuzzy logics." In: *Non-standard Logics for Automated Reasoning* (P. Smets et al. eds.), 287-325, Academic Press, 1988.
2. M. C. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Vol. 169 of Synthese Library, D. Reidel Publishing Company, 1983.
3. S. A. Kripke. "Semantical considerations on modal logic." *Acta Philosophica Fennica*, 16:83-94, 1963.
4. C. J. Liau and I. P. Lin. "Quantitative modal logic and possibilistic reasoning". *Proc. of ECAI92*, 43-47, 1992.
5. E. H. Ruspini. "On the Semantics of Fuzzy Logic." *International Journal of Approximate Reasoning*, 5:45-88, 1991.

Approximation Methods for Knowledge Representation Systems

Cecylia M. Rauszer

Institute of Mathematics, University of Warsaw
02-097 Warsaw, Banacha 2, Poland
e-mail: rauszer@mimuw.edu.pl

Abstract. In the paper we examin an approximation logic which may serve as a tool in investigations of distributive knowledge representation systems. The concept of knowledge is based on rough set approach. The logic under consideration is strong enough to capture properties of knowledge understood as partitions of the universe, lower and upper approximations of sets of objects as well as properties of dependencies of attributes.

1 Introduction

Reasoning about knowledge has long been an issue of concern in philosophy and artificial intelligence. A number of papers addressing the problems e.g. [1] [2], [4], [5] have confirmed the important role of this domain.

One of the difficulties which occur when we want to formally reason about knowledge is lack of an agreement what we mean by knowlegde or what properties knowledge satisfies or should satisfy. This problem is particulary important when we want to find out good semantics which allows us to reason about knowledge.

In this paper, we focus on rough-set approach ([6], [7]) to modeling knowledge. In our approach knowledge is understood as the ability of classification of objects. By objects we mean anything what a human being can think of. Objects create the universe of discourse U, called shortly universe. Any partition of U is said to be *knowledge about the universe* or simply *knowledge*. For instance, if $U = \{o_1, o_2, o_3, o_4, o_5\}$ then the partitions $\mathcal{E}_s = \{\{o_1, o_2, o_3\}, \{o_4\}, \{o_5\}\}$ and $\mathcal{E}_t = \{\{o_1, o_2\}, \{o_3, o_4\}, \{o_5\}\}$ are examples of knowledge about the universe U.

To give some intuitions of the problems discussed in the paper it is convenient to assume that each partition is viewed as a knowledge of agent t about the universe U.

One of the problems we are interested in is how to represent knowledge. Suppose that U is the universe of discourse and let $\mathcal{E}_t$ denote any knowledge about U. What we want to find out is a "real" description of objects which classifies them in the same way as $\mathcal{E}_t$.

For this purpose, we use so called information system [7]. An information system in our approach is in fact a data table with columns labelled by attributes and rows labelled by objects. Each row in the data table represents information

about the corresponding object. In general in a given information system we are not able to distinguish all single objects (using attributes of the system). Namely, objects can have the same values on some attributes or all attributes. As a consequence, any set of attributes divides the universe U on some classes which establish a partition of the set of all objects U. Suppose now, that $\mathcal{S}_t$ is an information system describing the universe U by means of a set of attributes $\mathbf{A}_t$. Denote it by $(U, \mathbf{A}_t)$, that is, $\mathcal{S}_t = (U, \mathbf{A}_t)$ and let $\mathcal{E}$ be the partition given by $\mathbf{A}_t$. If $\mathcal{E} = \mathcal{E}_t$ then we say that $\mathcal{S}_t$ represents knowledge $\mathcal{E}_t$ and call $\mathcal{S}_t$ a *knowledge representation system* of $\mathcal{E}_t$. Clearly, a such representation is not unique.

Now, let $\mathcal{E} = \{\mathcal{E}_t\}_{t\in T}$ be a family of partitions of U, that is, each $\mathcal{E}_t$ is considered as knowledge about U. Let $\{\mathcal{S}_t\}_{t\in T}$ be the family of knowledge representation systems for $\mathcal{E}$, that is, for every t, $\mathcal{S}_t$ is a knowledge representation system of $\mathcal{E}_t$. We will call the family $\{\mathcal{S}_t\}_{t\in T}$ *distributive knowledge representation systems* of $\mathcal{E}$. Notice, that each $\mathcal{S}_t$ may be viewed as a perception of the universe U by agent t.

Now, let $\mathcal{E}_s$ and $\mathcal{E}_t$ be two different classifications of objects from U. We define so called *strong common knowledge* of $\mathcal{E}_t$ and $\mathcal{E}_t$, denoted by $\mathcal{E}_{s\vee t}$, and *weak common knowledge*, denoted by $\mathcal{E}_{s\wedge t}$ of $\mathcal{E}_s$ and $\mathcal{E}_t$. Intuitively speaking, strong common knowledge is a partition of the universe such that a better recognition of a set of objects by one classification is preserved. In other words, better knowledge about objects is reflected in $\mathcal{E}_{s\vee t}$. If $\mathcal{E}_s$ is interpreted as knowledge of an agent s and $\mathcal{E}_t$ as knowledge of an agent t about the same universe U then $\mathcal{E}_{s\vee t}$ may be viewed as knowledge of a the group $\{s,t\}$, denoted by st, which intuitively may be described as follows: If an agent s knows that an object x is different from objects $x_{j_1}, \ldots, x_{j_m}$ then x has to be different from those objects in $\mathcal{E}_{s\vee t}$. If for an agent t, according to her knowledge about the universe U, the same object x differs from $x_{i_1}, \ldots, x_{i_n}$, then the objects x is perceived by group st as the object which is different from $x_{j_1}, \ldots, x_{j_m}, x_{i_1}, \ldots, x_{i_n}$. As a consequence we have: if an object x is distinguished from y in $\mathcal{E}_{s\vee t}$, that is, the block $[x]$ is different from the block $[y]$ (in $\mathcal{E}_{s\vee t}$) then it means that at least one of agents s, t distinguishes x from y. If objects x and y are indiscernible in $\mathcal{E}_{s\vee t}$, then it means that each agent from the group st, according to her knowledge, is not able to distinguish between x and y. For instance, in the above example the partition $\mathcal{E}_s$ better classifies object o_4 than knowledge $\mathcal{E}_t$. Using knowledge $\mathcal{E}_t$ objects o_3 and o_4 are indiscernible. Hence, the class $\{o_4\}$ has to appear in $\mathcal{E}_{s\vee t}$.

The notion of weak common knowledge may be interpredted as knowledge where weaker knowledge about objects dominates in $\mathcal{E}_{s\wedge t}$. In other words, if an agent s identifies x with y, that is, $\{x, y\}$ is a block in $\mathcal{E}_s$ then the group st also identifies x with y. If moreover t identifies y with z and distinguishes between x and y, that is, $\{y, z\} \in \mathcal{E}_t$, then it means that the block $\{x, y, z\}$ belongs to $\mathcal{E}_{s\wedge t}$. As a consequence we have that if $[x] = \{x\} \in \mathcal{E}_{s\wedge t}$, then it means that there is consensus of agents s and t about object x, i.e., $[x] \in \mathcal{E}_s \cap \mathcal{E}_t$. If $\{x_1, \ldots, x_n\} \in \mathcal{E}_{s\wedge t}$, then it means that either there is consensus of agents s and t about $x_1, \ldots, x_n$ or for any subset of $\{x_1, \ldots, x_n\}$ there is no consensus. For instance, because of the objects o_3 and o_4 are indiscernible by $\mathcal{E}_t$ they have

to remain indiscernible in $\mathcal{E}_{s\wedge t}$. Moreover, o_3 is not distinguished from objects o_1, o_2 by $\mathcal{E}_s$. Thus the block $\{o_1, o_2, o_3, o_4\}$ is an element of the partition $\mathcal{E}_{s\wedge t}$.

We have shown that the family $\mathcal{E} = \{\mathcal{E}_t\}_{t\in T}$ may be considered as a lattice. Moreover, $\mathcal{E}_{s\vee t}$ is the infimum of $\mathcal{E}_s$ and $\mathcal{E}_t$ in $\mathcal{E}$ and $\mathcal{E}_{s\wedge t}$ is the supremum of $\mathcal{E}_s$ and $\mathcal{E}_t$ in $\mathcal{E}$.

The operation of weak common knowledge seems to be more interesting than the operation of strong common knowledge. Namely, we show that if $\mathcal{S}_s = (U, \mathbf{A}_s)$ is a knowledge representation system of $\mathcal{E}_s$ and $\mathcal{S}_t = (U, \mathbf{A}_t)$ is a knowledge representation system of $\mathcal{E}_t$, then the system $(U, \mathbf{A}_s \cup \mathbf{A}_t)$ represents strong common knowledge of the group st i.e., $\mathcal{E}_{s\vee t} = \mathcal{E}_{\mathbf{A}_s\cup\mathbf{A}_t}$. In the case of weak common knowledge we have $\mathcal{E}_{s\wedge t} \neq \mathcal{E}_{\mathbf{A}_s\cap\mathbf{A}_t}$. Clearly, if $\mathcal{E}_{s\wedge t}$ is a weak common knowledge of $\mathcal{E}_s$ and $\mathcal{E}_t$ then there is a knowledge representation system $(U, \mathbf{A})$ such that the partition determined by $\mathbf{A}$ equals to $\mathcal{E}_{s\wedge t}$. However, the set of attributes $\mathbf{A}$ need not to be the same as the set $\mathbf{A}_s \cap \mathbf{A}_t$.

It is shown also that the family $\{\mathcal{S}_t\}_{t\in T}$ of distributed knowledge representation systems may be treated as a lattice. The intuitive meaning of the lattice ordering $\leq$ is that, if $\mathcal{S}_s \leq \mathcal{S}$ then the sharpness of perception of U and therefore feature recognitions of objects from the universe U by agent s is weaker than that of agent t.

In the paper we examin lower and upper approximations of sets of objects [8] in distributive knowledge systems. Intuitively speaking, by a lower approximation of X in $\mathcal{S} = (U, \mathbf{A})$ we mean the set of objects of U which without any doubt belong to X. An upper approximation of X is a set of objects which could be classified as elements of X. Finally, we consider boundary of X which is in a sense undecidable area of the universe.

Finally, we introduce and examin formal system called *approximation logic*. This logic is intended as a logic which reflects properties of knowledge (in our sense), properties of approximations and some other features of distributive knowledge representation systems. The idea of this logic is based on [Ra]. Roughly speaking, our logic is a modal logic with finite number of modal operators. Each modal operator corresponds to knowledge represented by an information system. A formula of the form $I_{\mathbf{A}}\phi$ may be interpreted as a lower approximation of the set of objects which satisfy ϕ in the information system $\mathcal{S} = (U, \mathbf{A})$.

2 Knowledge Base

In this paper, knowledge is understood as the ability to classify objects. By objects we mean anything a human being can think of, as for example, abstract concepts, real things, processes, states, etc. Objects are treated as elements of real or abstract world called the *universe of discourse* or shortly the *universe*. Hence, in our approach knowledge is strictly connected with classification of parts of the universe.

We explain this idea more precisely.

Let U be the universe of discourse. Any subset X of U is said to be a *concept*. Any family of concepts which forms a partition of U will be referred to us as ***knowledge about*** U, or shortly ***knowledge***.

Let R be an equivalence relation over U. By $\mathcal{E}_R$ we denote the family of all equivalence classes (blocks) of R. Thus, in our terminology $\mathcal{E}_R$ is knowledge about U and each block $[x]_R$ is considered as a concept. So, if R is an equivalence relation then R determines a classification $\mathcal{E}_R$ of objects from the universe U. If a block $[x]_R \in \mathcal{E}_R$ contains more than one object then objects from $[x]_R$ are not distinguishable with respect to knowlegde $\mathcal{E}_R$.

Let $\mathcal{R} = \{R_t\}_{t\in T}$, where T is a finite set, be a family of equivalence relations over U. By ***knowledge base*** we mean any relational system $\mathcal{K} = (U, \{R_t\}_{t\in T})$. Each R_t, where $t \in T$, determines knowledge $\mathcal{E}_{R_t}$ about U and each block $[x]_{R_t}$ from $\mathcal{E}_{R_t}$ is a concept called a *basic concept*. To simplify notation we will write $\mathcal{E}_t$ instead of $\mathcal{E}_{R_t}$ and $[x]_t$ instead of $[x]_{R_t}$, for any $t \in T$.

Let $(U, \{R_t\}_{t\in T})$ be a knowledge base such that $\{R_t\}_{t\in T}$ is the family of all equivalence relations on U and let $\mathcal{E} = \{\mathcal{E}_t\}_{t\in T}$ be a family of all partitions determined by $\{R_t\}_{t\in T}$. Let $\prec$ be a binary relation on $\mathcal{E}$ defined as follows:

$$\mathcal{E}_s \prec \mathcal{E}_t \text{ if and only if } \forall[x]_s \exists[y]_t \ [x]_s \subseteq [y]_t.$$

It might be proved that $\prec$ is an ordering relation on $\mathcal{E}$, that is, $(\mathcal{E}, \prec)$ is a poset.

Denote by $\mathcal{E}_s \wedge \mathcal{E}_t$ the following set:

$$\{[x]_s \cap [y]_t : [x]_s \cap [y]_t \neq \emptyset\}.$$

It might be easily proved that $\mathcal{E}_s \wedge \mathcal{E}_t$ is a partition of U and moreover, $\mathcal{E}_s \wedge \mathcal{E}_t$ is the *infimum* (*inf*) of $\mathcal{E}_s$ and $\mathcal{E}_t$ in $(\mathcal{E}, \prec)$, that is, for every $s, t \in T$, the $inf\{\mathcal{E}_s, \mathcal{E}_t\}$ exists in the poset $(\mathcal{E}, \prec)$. We call $\mathcal{E}_s \wedge \mathcal{E}_t$ a *strong common knowledge* of $\mathcal{E}_s$ and $\mathcal{E}_t$.

Now put

$$\mathcal{E}_s \vee \mathcal{E}_t = \{X \subseteq U : X \text{ is the union of } \textbf{all } [x]_s \text{ and } [y]_t \text{ such that } [x]_s \cap [y]_t \neq \emptyset\}.$$

It is easy to show that $\mathcal{E}_s \vee \mathcal{E}_t$ is a partition of U. Moreover, $\mathcal{E}_s \vee \mathcal{E}_t$ is the *supremum* (*sup*) in $(\mathcal{E}, \prec)$ of $\mathcal{E}_s$ and $\mathcal{E}_t$. Thus for any $\mathcal{E}_s$ and $\mathcal{E}_t$ in $\mathcal{E}$, the $sup\{\mathcal{E}_s, \mathcal{E}_t\}$ exists in the poset $(\mathcal{E}, \prec)$. We call the partition $\mathcal{E}_s \vee \mathcal{E}_t$ a *weak common knowledge* of $\mathcal{E}_s$ and $\mathcal{E}_t$.

Lemma 2.1 *$(\mathcal{E}, \vee, \wedge, \mathbf{0}, \mathbf{1})$ is a lattice with the zero element and the unit element.*

□

We call every sublattice of the lattice $(\mathcal{E}, \vee, \wedge, 0, 1)$ a *lattice of partitions*. One can show that a lattice of partition need not to be distributive.

We finish this section with the following remark. Let $\mathcal{E} = \{\mathcal{E}_t\}_{t\in T}$ be a knowledge base over U. It might be easily shown that the relation $\leq$ defined on the set of indices T as follows:

$$s \leq t \text{ if and only if } \mathcal{E}_t \prec \mathcal{E}_t.$$

is an ordering relation T. Thus $(T, \leq)$ is a poset. Moreover, if $\mathcal{E}$ is a lattice of partitions, that is, $\mathcal{E}$ is closed under the binary operations $\vee$ and $\wedge$ defined above and $inf\mathcal{E}$ and $sup\mathcal{E}$ belong to $\mathcal{E}$ then one can prove that $sup\{s,t\}$ denoted $s \vee t$ and $inf\{s,t\}$ denoted $s \wedge t$ exist in $(T, \leq)$. Namely, we have

$$s \vee t = sup\{s,t\} \text{ if and only if } \mathcal{E}_{s\vee t} = \mathcal{E}_s \wedge \mathcal{E}_t$$

and

$$s \wedge t = inf\{s,t\} \text{ if and only if } \mathcal{E}_{s\wedge t} = \mathcal{E}_s \vee \mathcal{E}_t.$$

3 Lower and Upper Approximation

Observe that some concepts may be expressed as the set-theoretical *union* of certain basic concepts from one knowledge about U but they cannot be defined as the union of basic blocks from another knowledge. Hence, if a concept cannot be covered by basic concepts from a given knowledge base $\mathcal{E}_t$ *then the question* arises whether it can be "approximately" defined by $\mathcal{E}_t$. In this section we are going to discussed this problem.

Let R be any equivalence relation over U. With every concept X, $X \subseteq U$, we associate three sets: $\underline{R}(X)$, $\overline{R}(X)$ and $B_R(X)$ called *R-lower approximation of X*, *R-upper approximation of X* and *R-boundary of X* , respectively, where

$$\underline{R}(X) = \{[x]_R : [x]_R \subseteq X\},$$
$$\overline{R}(X) = \{[x]_R : [x]_R \cap X \neq \emptyset\},$$

and

$$B_R(X) = \overline{R}(X) - \underline{R}(X).$$

Intuitively speaking the lower approximation of X is the collection of all elements of the universe which can be classified with full certainty, as elements of X, using knowledge R. The upper approximation of X is the collection of objects from the universe U which can be possibly classified as elements of X, using knowledge R. Finally, the boundary of X is in a sense undecidable area of the universe, that is, none of the objects belonging to $B_R(X)$ can be classified with certainty into X or $-X$ as far as knowledge R is concerned.

We say that a concept X is *R-definable* if $\underline{R}(X) = \overline{R}(X)$. If for some subset X of U, $\underline{R}(X) \neq \overline{R}(X)$ then X is said to be *R-rough*.

From now on we will assume that $\mathcal{E} = \{\mathcal{E}_t\}_{t\in T}$ is a lattice of partitions and we will consider T also as a lattice.

Now, let $\mathcal{R} = \{R_t\}_{t\in T}$ be the family of equivalence relations over U determined by $\{\mathcal{E}_t\}_{t\in T}$.

Because of the duality between lower and upper approximation we list only properties of lower approximation.

Lemma 3.1 *For every $t \in T$*

$$\mathcal{E}_t \prec \mathcal{E}_s \text{ if and only if } \underline{R}_s(X) \subseteq \underline{R}_t(X).$$

□

Lemma 3.2 *Let $\mathcal{R} = (U, \{R_t\}_{t\in T})$ be a knowledge base such that $\{R_t\}_{t\in T}$ is defined by $\{\mathcal{E}_t\}_{t\in T}$. For every equivalence relations R_s and R_t from $\mathcal{R}$ and every set $X \subseteq U$ the following hold:*

1. $\underline{R}_t(X) \cup \underline{R}_s(X) \subseteq \underline{R}_{t\vee s}(X)$.
2. $\underline{R}_{s\wedge t}(X) \subseteq \underline{R}_t(X) \cap \underline{R}_s(X)$.
3. $\underline{R}_t\underline{R}_s(X) \subseteq \underline{R}_t(X)$.
4. $\mathcal{E}_s \prec \mathcal{E}_t$ *implies* $\underline{R}_t\underline{R}_s(X) = \underline{R}_t(X)$.

□

Next lemma characterizes the boundary of X determined by any equivalence relation from $\mathcal{R}$.

Lemma 3.3 *Let $(U, \{R_t\}_{t\in T})$ be a knowledge base. For every $X \subseteq U$ the following conditions are true:*

1. $\mathcal{E}_t \prec \mathcal{E}_s$ *if and only if* $B_t(X) \subseteq B_s(X)$.
2. $B_{s\vee t}(X) \subseteq B_s(X) \cap B_t(X)$.
3. $B_s(X) \cup B_t(X) \subseteq B_{s\wedge t}(X)$.
4. $B_t(X) = B_t(-X)$.
5. $\underline{R}_t B_t(X) = B_t(X)$.
6. $B_t(X) = \emptyset$ *if and only if* $\underline{R}_t(X) = X$.

□

4 Information Systems

In this section we recall notion of information systems introduced by Pawlak [8].

An *information system* is a pair $\mathcal{S} = (U, \mathbf{A})$, where

U - a nonempty, finite set of objects, called the *universe of discourse* of $\mathcal{S}$,

$\mathbf{A}$ - a finite set of *attributes* i.e.

$a : U \to V_a$ for $a \in \mathbf{A}$,

where V_a is called *the value set of* a.

The set $V = \bigcup_{a\in\mathbf{A}} V_a$ is said to be a domain of $\mathbf{A}$.

For any object o from U, $o \mapsto (a, v)$ means that the object o has the value v for the attribute a. For instance, if an attribute a is *color*, its value v is *green* then $o \mapsto (color, green)$ means that object o is green in $\mathcal{S}$.

Instead of $o \mapsto (a, v)$ we will also write $a(o) = v$.

With every subset of attributes $B \subseteq \mathbf{A}$ we associate a binary relation $ind(B)$, called *indiscernibility relation*, and defined as follows:

$$ind(B) = \{(x, y) \in U \times U : \forall a \in B\ a(x) = a(y)\}\ .$$

Notice that objects x, y satisfying the relation $ind(B)$ are indiscernible with respect to the attributes from B. In other words, the information system $\mathcal{S}$ does not distiguish x from y in terms of attributes in B. By $[o]_B$ we denote the equivalence class of $ind(B)$ including the object o, i.e. the set $\{y \in U : o\ ind(B)\ y\}$. Clearly, for every $B \subseteq \mathbf{A}$ the family of all equivalence classes of the relation $ind(B)$ is a partition of U. Denote this partition in the following way:

$$\mathcal{E}_B = \{[o_1]_B, \ldots, [o_n]_B\}.$$

Let $\mathcal{E} = \{\mathcal{E}_B\}_{B \subseteq \mathbf{A}}$. Clearly $\mathcal{E}$ is a lower semi-lattice. One can show that $\mathcal{E} = \{\mathcal{E}_B\}_{B \subseteq \mathbf{A}}$ need not to be a lattice of partitions.

Because of each indiscernibility relation is an equivalence relation we can express notion of lower and upper approximation from Section 3 in terms of information systems.

Let an information system $\mathcal{S} = (U, \mathbf{A})$ be given. For every $B \subseteq \mathbf{A}$, $\underline{ind}(B)(X)$ is a lower approximation of X and $\overline{ind}(B)(X)$ is an upper approximation of X in $\mathcal{S}$. To simplify notation we will denote these sets by $\underline{B}(X)$ and $\overline{B}(X)$, respectively and call the B-lower approximation and the B-upper approximation of X, respectively. A set $X \subseteq U$ is said to be *B-definable* if and only if $\underline{B}(X) = \overline{B}(X)$ and X is called *B-rough* if $\underline{B}(X) \neq \overline{B}(X)$.

5 Knowledge Representation Systems

It turns out that every knowledge base may be represented as an information system, that is, in the form of an attribute-value table.

Let $\mathcal{E}_t$ be a knowledge about U. It is not difficult to construct an information system $\mathcal{S}_t = (U, \mathbf{A}_t)$ such that $\mathcal{E}_t = \mathcal{E}_{\mathbf{A}_t}$. We call an information system $\mathcal{S}_t = (U, \mathbf{A}_t)$ a *knowledge representation system* (*k.r.s.*) for $\mathcal{E}_t$ provided $\mathcal{E} = \mathcal{E}_{\mathbf{A}_t}$. If $\mathcal{K} = (U, \{R_t\}_{t \in T})$ is a knowledge base, then the family $\mathcal{S} = \{\mathcal{S}_t\}_{t \in T}$ is said to be a *distributive knowledge representation system* of $\mathcal{K}$ provided for every $t \in T$, $\mathcal{S}$ is a *k.r.s.* of R_t.

Observe, that if $\mathcal{S} = \{\mathcal{S}_t\}_{t \in T}$ is a distributive knowledge representation system of $\mathcal{K} = (U, \{R_t\}_{t \in T})$ then the relation $\leq$ defined in Section 2 may be viewed as a relation between knowledge representation systems. Namely, the intuitive meaning of the relation $\leq$ is, that if $s \leq t$ than the feature recognitions of objects from U by *k.r.s.* $\mathcal{S}_s = (U, \mathbf{A}_s)$ is weaker than that of $\mathcal{S}_t = (U, \mathbf{A}_t)$. In other words, the classification of objects from the universe U by means of the set of attributes $\mathbf{A}_t$ is better than the classification of these objects by means of $\mathbf{A}_s$. If $s \leq t$ then we say that *k.r.s.* $\mathcal{S}_s$ is less efficient or weaker than the system $\mathcal{S}$.

Let $\mathcal{K}$ be a knowledge base and let $\mathcal{S} = \{\mathcal{S}_t\}_{t \in T}$ be a distributive knowledge representation system of $\mathcal{K}$. Notice that $(U, \mathbf{A})$, where $\mathbf{A} = \bigcup_{t \in T} \mathbf{A}_t$ has the following property: for every $t \in T$ there is a set $B \subseteq \mathbf{A}$ such that $ind(B) = R_t$. Indeed, it follows from the properties of indiscernibility relation (see Section 4) that if $\mathcal{S}_s = (U, \mathbf{A}_s)$ and $\mathcal{S}_t = (U, \mathbf{A}_t)$ are knowledge representation systems of $\mathcal{E}_s$ and $\mathcal{E}_t$ respectively, then knowledge representation system of the strong common

knowledge $\mathcal{E}_{svt}$ is the information system of the form $(U\ \mathbf{A}_s \cup \mathbf{A}_t)$. Hence, for every knowledge base $\mathcal{K} = (U, \{R_t\}_{t\in T})$ there is an information system $\mathcal{S} = (U, \mathbf{A})$ such that for every $t \in T$ there is a set $B \subseteq \mathbf{A}$ for which $ind(B) = R_t$. We call also such a system a *knowledge representation system* for $\mathcal{K}$.

It is worth while to observe that for every information system $\mathcal{S} = (U, \mathbf{A})$ there is a knowledge base $\mathcal{K} = (U, \{R_t\}_{t\in T})$ such that for every $B \subseteq \mathbf{A}$ there is an equivalence relation R_t such that $ind(B) = R_t$. Namely, take as $\mathcal{K}$ the system $(U, \{ind(B)\}_{B\subseteq \mathbf{A}})$.

From now on the notion of knowledge representation system will be used as a synonymous with information system.

6 Approximation Logic

6.1 Syntax

In this section we are going to define a logic which may be used as a logical tool in the investigations of distributive knowledge representation systems. We want to construct a formal system which allows us to formalize relations between knowledge representation systems and which will describe lower and upper approximation of any concept as well as boundary region of concepts.

We call this logic *approximation logic* and denote A-logic. Let us emphesize that A-logic defined here is not related to approximation logics defined and examined in [10].

Let $\mathbf{A}$ be a finite set of attributes and let $V = \bigcup_{a\in\mathbf{A}} V_a$ be a family of finite sets. Each V_a may be treated as a domain of an attribute $a \in \mathbf{A}$.

We associate with A and V a language $\mathcal{L}_{\mathbf{A},V}$ called *the formal language of A-logic*. All subsets of $\mathbf{A}$ and elements of V are treated as constants of $\mathcal{L}_{\mathbf{A},V}$. Subsets of $\mathbf{A}$ are called *attribute constants* and denoted by $a, b, c, \ldots$ and $A, B, C, \ldots$. Elements of V are called *attribute-value constants* and denoted by $v, u \ldots$.

The language $\mathcal{L}_{\mathbf{A},V}$ consists of two levels. The expressions of these levels are called *formulae of the 1-st kind* and *formulae of the 2-nd kind* , respectively. Intuitively, formulae of the 1-st kind describe properties of knowledge understood as a partition of the universe U, whereas formulae of the 2-nd kind express certain facts concerning sets of objects of U and approximation of these sets.

To give a formal definition of the sets of formulae we define first terms of $\mathcal{L}_{\mathbf{A},V}$.

Terms are built up from attribute constants, two constants $\mathbf{0}$ and $\mathbf{1}$ and operations: $\vee$ and $\wedge$. More precisely, the *set of all terms* is defined to be the least set T with the following three properties:

- $\mathbf{0}$ and $\mathbf{1}$ are in T.
- all attribute constants are in T,
- $B \vee C$, $B \wedge C$ are in T, whenever B, C are terms.

Formulae of the 1-st kind are built up from terms and two predicates $\leq$ and $=$. The set F_1 of all *formulae of the 1-st kind* is the smallest set such that

- If $B, C \in T$ then $B \leq C \in F_1$ and $B = C \in F_1$.

Intuitively, formulae of the 1-st kind express power of knowledge. Namely, $B \leq C$ may be interpreted in the following way: knowledge $\mathcal{E}_C$ is weaker than knowledge $\mathcal{E}_B$.

The set F_2 of all *formulae of the 2-nd kind* is the smallest set containing all atomic formulae which are of the form (a, v), where $a \in \mathbf{A}$ and $v \in V$ and it is closed with respect to propositional connectives $\vee$, $\wedge$, $\rightarrow$, $\rightarrowtail$, $\neg$ and the family $\{I_B\}_{B \subseteq \mathbf{A}}$ of modal connectives. For every B, I_B is called necessity operator.

Axioms for $\mathcal{A}$-logic consist of three groups: one for formulae of the 1-st kind , one for formulae of the 2-nd kind and one contains specific axioms for $\mathcal{A}$-logic. Axioms of the last group express characteristic properties of knowledge representation systems.

Axioms for formulae of the 1-st kind are as follows:

(t_1) All axioms for equality and for ordering relation,

(t_2) $B \leq B \vee C \qquad C \leq B \vee C$,

(t_3) $B \wedge C \leq B \qquad B \wedge C \leq C$,

where B and C are any terms.

As axioms for formulae of the 2-nd kind we assume all axioms for classical logic enriched by axioms for necessity operator.

The specific axioms of $\mathcal{A}$-logic are as follows:

1. $(a, v) \wedge (a, u) \rightarrowtail \bot$ for any $a \in \mathbf{A}$, $v, u \in V_a$ and $v \neq u$.
2. $\bigvee_{v \in V_a} (a, v) \rightarrowtail \top$, for every $a \in \mathbf{A}$.
3. $\neg(a, v) \rightarrowtail \bigvee\{(a, u) : u \in V_a, u \neq v\}$, for every $a \in \mathbf{A}$.

where $\bigvee \phi$ means a finite disjunction.

The motivation of the axioms above is the following: Specific axioms should be characteristic for our notion of knowledge representation system.

Observe that the first axiom follows from the assumption that each object can have exactly one value for each attribute.

Axiom (2) follows from the assumption that each object in any knowledge representation system $\mathcal{S}$ has a value *with respect to every attribute*. Hence, the description of objects is complete up to a given set of attributes. In other words, for every $a \in \mathbf{A}$ and every object x the entry in the row x and the column a (in $\mathcal{S}$ viewed as a table) is nonempty.

The third axiom allows us to figure out negation in such a way that instead of saying that an object does not possess a given property we can say that it has one of the remaining properties. For example, instead of saying that something is not blue we can say it is either red or green or yellow, etc.

We say that a formula ϕ is *derivable* in $\mathcal{A}$-logic from a set of formulae F, denoted by $F \vdash \phi$ provided it can be conclude from F by means of the above axioms and the following rules: modus ponens, and for every term B, C and D

$$\frac{\phi}{I_B \phi} \ ,$$

$$\frac{B \leq C}{I_B\phi \rightarrow I_C\phi}$$

$$\frac{B \leq D \quad C \leq D}{B \vee C \leq D} \qquad \frac{D \leq B \quad D \leq C}{D \leq B \wedge C}$$

If ϕ is derivable from the empty set, then we write $\vdash \phi$ and say ϕ *is derivable*. Clearly, all classical tautologies are derivable. Also $\vdash \top$ and $\not\vdash \bot$. Also we have e.g., $\vdash (I_B\phi \vee I_C\phi) \rightarrow I_{B\vee C}\phi$.

6.2 Semantics

Intuitively speaking, formulae of A-logic are meant as descriptions of objects of the universe and as connections between partitions of the universe. Formulae describe subsets of objects obeying properties expressed by these formulae. For instance, a natural interpretation of an atomic formula (a, v) is the set of all objects having value v for the attribute a. Hence, a natural interpretation of a formula of the form $I_B(a, v)$ is the B-lower approximation of the set of all objects having the property v for the attribute a.

Now, we are going to give semantics for the language of A-logic. Let $\mathcal{L}_{\mathbf{A},V}$ be the language described above determined by $\mathbf{A}$ and $V = \bigcup_{a\in\mathbf{A}} V_a$. Let $\mathcal{S} = (U, \mathbf{A})$ be an information system with the universe U, $|U| \geq 2^{\mathbf{A}}$ the set of attributes $\mathbf{A}$ and domain of attributes V. Moreover, let the family $\mathcal{E} = \{\mathcal{E}_B\}_{B\subseteq\mathbf{A}}$ of all partitions U determined by $\mathcal{S}$ be the lattice of partitions. Recall that $\mathcal{E}_B = \{[x_1]_B, \ldots, [x_l]_B\}$ is a partition of the universe U given by $ind(B)$.

We will interpret terms as partitions of the universe U. Namely, if B is a term then we interpret it as the partition $\mathcal{E}_B$ of U. The predicate $\leq$ is interpreted as the converse relation to $\prec$. Thus if $B \leq C$ is a formula of the 1-st kind, then we interpret it as $\mathcal{E}_C \prec \mathcal{E}_B$. Clearly, the interpretation of a formula of the form $B = C$ is, that, the partitions $\mathcal{E}_B$ and $\mathcal{E}_C$ of U are the same.

We say that *a formula ϕ of the 1-st kind is true in $\mathcal{S}$*, denoted by $\models \phi$, if the corresponding ralation holds in the lattice of partitions of $\mathcal{S}$. More precisely, a formula of the form $B \leq C$ (or $B = C$) is true in $\mathcal{S}$ if $\mathcal{E}_C \prec \mathcal{E}_B$ (or $\mathcal{E}_B = \mathcal{E}_C$) holds in the lattice $\mathcal{E}$.

Let ϕ be a formula of the 2-st kind from $\mathcal{L}_{\mathbf{A},V}$.

We say that an object $x \in U$ satisfies an atomic formula (a, v) if and only if $a(x) = v$. In the standard way we extend this definition on the set of all formulae of the 2-nd kind. If x satisfies ϕ then we will write $x \models \phi$.

Finally, we say that a formula ϕ of the 2-nd kind is *true in $\mathcal{S}$*, denoted by $\models \phi$, if $|\phi|_{\mathcal{S}} = \{x : x \models \phi\} = U$. If any formula ϕ (of the 1-st or of the 2-nd kind) is true in $\mathcal{S}$ then we call the information system $\mathcal{S}$ a *model* for ϕ.

We say that F *implies* ϕ, denoted by $F \models \phi$, if from the fact that $\mathcal{S}$ is a model of F follows that $\mathcal{S}$ is a model of ϕ.

Next two theorems show that our axiomatization has been adequatly choosen and that it is complete.

Theorem 6.1 *(soundness) Let F be a set of formulae. If $F \vdash \phi$ then $F \models \phi$.*

□

Theorem 6.2 *(completeness) Let F be a set of formulae and let ϕ be a formula of $\mathcal{L}_{\mathbf{A},V}$. If $F \models \phi$ then $F \vdash \phi$.*

□

For every term B define now new modal connectives C_B and Br_B as follows: for any formula of the 2-nd kind ϕ put:

$$C_B\phi \equiv \neg I_B \neg \phi$$

and

$$Br_B\phi \equiv C_B\phi \wedge \neg I_B\phi.$$

It is easy to check that

$$|C_B\phi|_{\mathcal{S}} = \overline{B}(|\phi|_{\mathcal{S}}) \ .$$

where C_B is the unary connective defined above, and $\overline{B}$ is upper approximation determined by $ind(B)$ in the knowledge representation system $\mathcal{S} = (U, \mathbf{A})$.

Moreover,

$$|Br_B\phi|_{\mathcal{S}} = B_B(|\phi|_{\mathcal{S}}),$$

where $B_B(|\phi|_{\mathcal{S}})$ means the boundary of $|\phi|_{\mathcal{S}}$ in $\mathcal{S}$ determined by $ind(B)$.

Theorem 6.3 *Let a knowledge representation system $\mathcal{S} = (U, \mathbf{A})$ be given. The following conditions are equivalent:*

1. *A set $X \subseteq U$ is B-definable in $\mathcal{S}$.*
2. *A formula of the form $I_B\phi - \phi$ is true in $\mathcal{S}$, where the meaning of ϕ is X, that is, $|\phi|_{\mathcal{S}} = X$.*
3. *$Br_B\phi$ is false, where ϕ is as before.*
4. *$|I_B\phi|_{\mathcal{S}} = X$.*

□

Finally we have

Theorem 6.4 *An information system $\mathcal{S} = (U, \mathbf{A})$ is a model for a formula ϕ if and only if there is a subset B of $\mathbf{A}$ such that the information system (U, B) is a model for ϕ.* □

References

1. Halpern, J. (ed.) (1986) *Proceedings of the Conference Theoretical Aspects of Reasoning about Knowledge*. Morgan Kaufmann.
2. Halpern J.Y., Moses Y. (1992) A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54, pp 309 - 379.
3. Hintikka, J.(1962) Knowledge an Belief. *Cornell University Press.*
4. Holland, J.H., Holyoak, K.J., Nisbett, R.E., and Thagard, P.R. (1986). Induction: Processes of Inference, Learning and Discovery. *MIT Press.*
5. Minski, M. (1975), A Framwork for Representation Knowledge. In *Winston, P. (ed) The Psychology of Computer Vision* McGraw-Hill,New York, pp 211-277.
6. Parikh, R (ed.) (1990) *Proceedings of the Conference Theoretical Aspects of Reasoning about Knowledge*, Morgan Kaufmann.
7. Pawlak, Z. (1982), Rough Sets. *International Journal of Computer and Information Sciences*, 11, pp. 341-346.
8. Pawlak, Z. (1991), Rough Sets- Theoretical Aspects of Reasoning about Data. *Kluwer.*
9. Pawlak, Z. (1990) Decision Logic, *Report of the Warsaw University of Technology, Institute of Computer Science.*
10. Rasiowa, H. (1990) On Approximation Logics: A survey. *Jahrbuch 1990 der Kurt Gödel Gesellschaft.*
11. Rauszer, C.M., Logic for Information Systems. to appear in *Fundamenta Informaticae.*
12. Skowron, A., On Topology in Information Systems. Bull of the Polish Academy of Sciences, vol 36, No 7-8, 1988, pp.477-479.

Modelling of Industrial Systems

Lennart Ljung
Department of Electrical Engineering, Linköping University
S-581 83 Linköping, Sweden

Abstract

In this contribution we give an overview over those techniques – mainly from the control field - that are used to derive models of dynamical systems from observed data.

1 Main stream System Identification

A typical problem

Here is a typical system identification problem: We observe inputs $u(t)$ and outputs $y(t)$ from a system. Here t denotes sample points: $t = 1, \ldots, N$. We want to construct a model of the system, and may seek a model of the simple form

$$y(t) + ay(t-1) = b_1 u(t-1) + b_2 u(t-2) \tag{1}$$

It thus just remains to determine suitable values of the parameters a, b_1 and b_2. This could be done by the well known *least squares* method.

$$\min_{a,b_1,b_2} \sum_{t=1}^{N} (y(t) + ay(t-1) - b_1 u(t-1) - b_2 u(t-2))^2 \tag{2}$$

The minimizing values $\hat{a}^N$, $\hat{b}_1^N$ and $\hat{b}_2^N$ can in this case be easily computed since (2) is a quadratic function. They give the model

$$y(t) + \hat{a}^N y(t-1) = \hat{b}_1^N u(t-1) + \hat{b}_2^N x(t-2) \tag{3}$$

of the system. This simple system identification problem is a special case of a broad class of model building problems, which we now describe:

Training sets and mathematical models

Here is an archetypical problem in science and human learning:"We are shown a collection of vector pairs $\{[y(t); x(t)]; t = 1, \ldots N\}$. Call this "the training set". We are then shown a new value $x(N+1)$ and are asked to name a corresponding value $y(N+1)$." The variable t could be thought of as time, but could be anything. The vectors $y(t)$ and $x(t)$ may take values in any sets (finite sets or subsets of $\Re^n$ or anything else) and the dimension of $x(t)$ could very well depend on t (and could be unbounded). The formulation covers most kinds of classification and model building problems.

How to solve this problem? The mathematical modelling approach is to construct a function $\hat{g}_N(t, x(t))$ based on the "training" set, and to use this function for pairing $y(t)$ to new $x(t)$:

$$\hat{y}(t) = \hat{g}_N(t, x(t)) \tag{4}$$

Where do we get the function g from? Essentially we have to search for it in a family of functions that is described (parametrized) in terms of a finite number of parameters. These parameters will be denoted by θ. The family of candidate model functions will be called a *model structure*, and we write the function as

$$g(t, \theta, x(t)) \tag{5}$$

The value $y(t)$ is thus matched against the "candidate" $g(t, \theta, x(t))$:

$$y(t) \sim g(t, \theta, x(t)) \tag{6}$$

We shall also use the notation

$$\hat{y}(t|\theta) = g(t, \theta, x(t)) \tag{7}$$

to stress that g is a "predicted" or "guessed" y-value. The search for a good model function is then carried out in terms of the parameters θ, and the chosen value $\hat{\theta}_N$ gives us

$$\hat{g}_N(t, x(t)) = g(t, \hat{\theta}_N, x(t)) \tag{8}$$

The case (1) corresponds to

$$\begin{aligned} \theta &= [a, b_1, b_2] \\ x(t) &= [y(t-1), u(t-1), u(t-2)] \\ g(t, \theta, x(t)) &= -ay(t-1) + b_1 u(t-1) + b_2 u(t-2) \end{aligned} \tag{9}$$

In general the function g is a mapping from the set where $x(t)$ takes its values to the space where $y(t)$ takes its values. All kinds of parameterizations are possible, from ones that are tailor-made for the application to general orthogonal functions expansion and neural net structures.

Signals and Dynamical Systems

The general formulation above fits into conventional system identification, which corresponds to particular functions g. We have already shown that the simple (ARX) model (1) fits into the framework.

In general the task to form pairs $[y(t), x(t)]$ is the one-step ahead prediction problem. For example, first order ARMA model of $\{y(t)\}$

$$y(t) + ay(t-1) = e(t) + ce(t-1) \tag{10}$$

where $\{e(t)\}$ is a white noise sequence, is obtained for

$$\theta = [a \quad c]; x(t) = [y(0), y(1) \ldots y(t-1)]$$

$$g(t, \theta, x(t)) = \sum_{k=0}^{t-1} (c-a)(-c)^{t-k-1} y(k) \tag{11}$$

etc. Fuzzy - or verbal - dynamical models can be obtained if $x(t)$ and $y(t)$ take on values like "the oven is very hot", "the oven is warm", "the water is boiling" and so on. The function g would then be some kind of a table - perhaps implemented in an expert system shell - and its parameters θ would describe the structure of the table.

Fitting model structures to data

The leading principle for choosing θ clearly is to have $g(t, \theta, x(t))$ perform well on the training set, that is to make

$$y(t) \text{ close to } g(t, \theta, x(t)) \; t = 1, \ldots, N \tag{12}$$

This principle applies also to the case where x and y assume non-numeric values, if only "close" can be appropriately defined. Most numeric schemes select $\theta = \hat{\theta}_N$ so that

$$\sum_{t=1}^{N} \| \, y(t) - g(t, \theta, x(t)) \, \| \tag{13}$$

is minimized for some norm $\| \cdot \|$ ("norm" should here be taken in a broad sense) or so that $y(t) - g(t, \theta, x(t))$ is uncorrelated with information in $x(t)$.

Typical asymptotic properties

A key question is: How good are the estimates obtained by (13)? The typical analysis goes as follows: Suppose that the pairs $[y(t), x(t)]$ really are related by

$$y(t) = g_0(t, x(t)) + v(t) \tag{14}$$

where $\{v(t)\}$ is an as yet undefined sequence

- **Consistency**: Suppose there is a "true" system description available within the model structure. We translate that as for some θ_0, $g(t, \theta_0, x(t)) = g_0(t, x(t))$ and $v(t)$ is white noise. Then $\hat{\theta}_N$ will converge to θ_0 as N increases to infinity, and the difference $\sqrt{N}(\hat{\theta}_N - \theta_0)$ will converge in distribution to a Gaussian random variable (i.e. $\hat{\theta}_N$ tends to θ_0 with the "rate" $\sim 1/\sqrt{N}$)

- **Convergence**: Suppose no "true" description is available in the model structure, but assume that $\{v(t)\}$ in (14) is white noise. Then $\hat{\theta}_N$ will converge to a value θ_* such that $g(t, \theta_*, x(t))$ approximates $g_0(t, x(t))$ as well as possible in the chosen norm in (13). Moreover $\sqrt{N}(\hat{\theta}_N - \theta_*)$ converges in distribution to a Gaussian random variable.

- **Making a sieve finer and finer** An interesting particular case is when the true system is assumed to belong to a very broad class of models, that cannot be parametrized by a finite number of parameters. However this class can be thought of as "the limit" of increasing model structures, that are parametrized by more and more parameters. (Think e.g. of an infinite dimensional system that can be seen as the limit of finite impulse response models as the number of coefficients tends to infinity). Mathematically this can be written as

$$g_0(t, x(t)) \text{belongs to } cl(\cup_{d=1}^{\infty} g_d(t, \theta^d, x(t)) \tag{15}$$

where the vector θ^d contains d parameters. To deal with this case it is customary to employ more and more parameters as more and more data becomes available. That is d becomes a function of N : $d(N)$.

If $\{v(t)\}$ in (14) is white noise and if $d(N)$ is chosen to increase to infinity slowly enough with N, we then have that the model will approach the true system as the number of data tends to infinity. This can be written formally as

$$\hat{g}_{d(N)}(t, \hat{\theta}_N^{d(N)}, x(t)) \longrightarrow g_0(t, x(t)) \text{ as } N \to \infty \tag{16}$$

With this we conclude our brief exposé of the main stream system identification. See the textbooks, e.g. [6] and [10] for the details of the general methods and results. Of course, system identification covers many other topics like how to compute the parameter values that minimize (13) and how to select the data so that they are as informative as possible.

2 The Model Structure

The single most important step in the identification process is to decide upon a model structure such as (6). In practice typically a whole lot of them are tried out and the process of identification really becomes the process of evaluating and choosing between the resulting models in these different structures.

It is natural to distinguish between three types of model structures.

1. Black-box structures
2. Structures from physical modelling
3. Structures from semi-physical modelling

2.1 Black-box structures

A black-box structure is one where the parametrization in terms of θ is chosen so that the family of models $\{g(t, \theta, x(t))\, \theta \in D_m\}$ covers as "many common and interesting" ones as possible. No particular attention to the actual application is then paid. For a linear system (a linear mapping from past data to future ones) we could for example think of choosing the parameters as the impulse response coefficients, of a finite impulse response model

$$\hat{y}(t|\theta) = \sum_{k=1}^{M} \theta_k u(t-k) \tag{17}$$

More common in control applications is the ARX black box structure for linear systems:

$$\hat{y}(t|\theta) = -a_1 y(t-1) - a_2 y(t-2) - \ldots - a_n y(t-n) + b_1 u(t-1) + \ldots + b_m u(t-m) \tag{18}$$

"the mother of all dynamical model structures".

In general we can write a black box structure conceptually as

$$\hat{y}(t|\theta) = \sum_{k=1}^{M} \theta_k h_k(x(t)) \tag{19}$$

i.e. as some kind of function expansion. In the general case the basis functions $\{h_k\}$ may also depend on θ.

It is instructive to distinguish between two principally different basis functions:

- Global: Each of the h_k have support in the whole x-space
- Local: Each of the h_k has support only in a small local box in the x-space.

Among black-box structures that use global basis functions are all the usual linear black box models, Volterra series expansions and so on.

The local basic functions models can be visualized as a multidimensional table: The x-space has been split up into a number of boxes. A new observation $x(t)$ then falls into one of these boxes, the one corresponding to say h_k, and the predicted output is then taken as θ_k (or possibly interpolated, taking into account few neighboring boxes). The sizes and locations of the boxes can be determined with the aid of estimation data. The extreme case is when the boxes are determined so that exactly one data point $x(t)\, t = 1 \ldots, N$ has fallen in each box: this is the so called nearest neighbor approach [13]. All this is well established in the statistical literature under names of "non-parametric regression" and "density estimation" e.g., [11], [2].

Neural network model structures, [8], represent a spectacular revival of these techniques. So called radial basis networks correspond to localized bases (where the "boxes" overlap like Gaussian distribution functions), while the feed-forward sigmoid network formally would use global basis functions (although the "dynamic effects" really are localized). Fuzzy modelling [5] is again an example of localized basis functions with typically polynomial interpolation rules, which are inherited by the "membership function".

It is worth stressing that these new techniques of neural net modelling and fuzzy identification represent useful revitalization of non-linear black box modelling with some new particular structures, but at the same time they definitely fall into a very old and classical framework of estimation techniques (See, e.g. [7], [1].)

2.2 Structures from physical modelling

In case we have physical insight into the properties of the system to be identified, it is natural to exploit this: *"Don't estimate what you already know!"* Basically we then write down those physical laws and relationships that describe the system. Most often they are then summarized in

a state space form like

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), \theta, u(t), v(t)) \\ y(t) &= h(t, x(t), \theta, v(t)) \end{aligned} \tag{20}$$

where θ denotes unknown physical constants in the description. The identification process is then to estimate these constants. That route takes us from (20) via (7) (explicitly or implicitly) and (12) to the estimate $\hat{\theta}_N$. The work to arrive at (20) and then to actually carry out the minimization of (12) can be considerable, though.

2.3 Semiphysical model structures

The logical route to utilize available physical knowledge may - as pointed out - be quite laborious. It is then tempting to instead try some simple black-box structures, such as the ARX model (18) *("Try Simple Things First")*. This is quite OK, but it should in any case be combined with physical insight. Here is a toy example to illustrate the point:

"Suppose we want to build a model for how the voltage applied to an electric heater affects the temperature of the room. Physical modelling entails writing down all equations relating to the power of the heater, heat transfer, heat convection and so on. This involves several equations, expressions and unknown heat transfer coefficients and so on. A simple black-box approach would instead be to use, say the ARX-model (18) with u as the applied voltage and y the room temperature. But that's too simple! A moment's reflection reveals that it's the heater power rather than the voltage that gives the temperature change. Thus use (18) with u= squared voltage and y= room temperature."

I would like to coin the term *semi-physical modelling* for introducing non-linear transformation of the raw measurement, based on high-school physics and common sense. The transferred measurements are then used in black-box structures such as the ARX structure.

Clearly semi-physical modelling is in frequent use. It is however also true that many failures of identification are indeed to be blamed on not applying this principle.

2.4 Hybrid structures

Of particular current interest is to conceive model structures that are capable of dealing both with dynamic effects, described by differential//-difference equations and with logical constraints, "the if:s and the but:s" of the system. Not so many concrete results have yet been obtained in this area, but quite intense work intense work is going on now. We may point to some work on using three models and pattern recognition for these hybrid model structures: [12], [9].

3 Model validation

It is not enough to come up with a nominal model $\hat{\theta}_N$ from (13) - we must also have a measure of its reliability. *Model validation* is the process of examining the model, assessing its quality and possibly rejecting its use for the purpose in question. In a sense this could be viewed as the *essential process of identification* - the estimation phase is really just a means to provide candidate models that might pass the needle's eye of validation.

Model validation has at least these different objectives:

1. To decide if the model is "good enough" for the intended application
2. To decide how "far from the true system description" the model might be
3. To decide whether the model and the data indeed are consistent with assumptions of the model structure.

These objectives partly overlap, but it is still possible to single out basic techniques:

1. The most obvious and pragmatic way to decide if a model is good enough is to test how well it is able to reproduce validation data (data that were not used to estimate the model) in simulation or

prediction. The user can then by eye inspection decide if the fit is "good enough". In my mind this is the prime validation tool.

2. To determine *error bounds* – how far is the true system from the model – is a fundamentally difficult question. If we adopt a probabilistic setting and assume that the true system is to be found within the chosen structure it becomes a matter to see how much the stochastic disturbances might have affected the model. The covariance matrix of the asymptotic distribution is classically used for the error bounds in this case. This covariance matrix is generally given by

$$cov\{\hat{\theta}_N\} \sim \frac{1}{N} Ev^2(t)[cov\{\frac{d}{d\theta}\hat{y}(t|\theta)\}]^{-1} \qquad (21)$$

for the structure (7), (14). If we (according to 3) below) cannot disprove that the true system can be represented in the chosen structure it is still reasonable to use the measure (21).

The remaining cases: No probabilistic setting adapted and/or the used model structure is known to be too simple has spurred a considerable interest recently [4]. It would lead too far to review that literature here.

3. The test if the data and the model are consistent with the model structure assumptions, is again a more straightforward task. Basically we compute the residuals $y(t) - \hat{y}(t|\hat{\theta}_N) = \epsilon(t)$ from the model and a (validation) data set and check if

   (a) $|\epsilon(t)| < C$ in a deterministic setting.

   (b) $\epsilon(t)$ and $u(t-\tau)$ are independent random variables, in a probabilistic setting (u is the input to the system).

   The latter test is one of many *residual* analysis tests that can be performed, and this is standard statistical practice, see e.g. [3].

4 Conclusions

We have in this contribution pointed to some basic issues in how to build mathematical models of real-life dynamical systems. The situation is well consolidated for purely dynamical systems, i.e. those that can be

described by difference or differential equation. The basic principles, efficient algorithms and well-spread commercial software purchases are well established. These techniques have also been in industrial use for quite some time.

Of great current interest is to move on to systems that also are characterized by logical constraints, switching dynamical properties depending on certain logical conditions and so on. This will probably require joint efforts between the control and computer science communities.

References

[1] A.R. Barron. Statistical properteis of artificial neural networks. In *Proceedings of the 28th IEEE Conference on Decision and Control*, pages 280–285, 1989.

[2] L. Devroye and L. Györfi. *Non-parametric density estimation.* Wiley, New York, 1985.

[3] N.R. Draper and H. Smith. *Applied Regression Analysis, 2nd ed.* Wiley, New York, 1981.

[4] G.C. Goodwin, M. Gevers, and B. Ninness. Quantifying the error in estimated transfer functions with application to model order selection. *IEEE Trans. Automatic Control*, 37(7):913–929, 1992.

[5] C.W. Ku and Y.Z. Lu. Fuzzy model identification and self-learning. *IEEE Trans. on SMC*, 17, 1987.

[6] L. Ljung. *System Identification - Theory for the User.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[7] L. Ljung and J. Sjöberg. A system identification perspective on neural nets. In S.Y. Kung et al, editor, *Neural Networks for Signal Processing, Proc of the 1992 IEEE-SP Workshop*, pages 423–435. IEEE Press, 1992.

[8] K.S. Narendra and K. Parathasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, 1:4–27, 1990.

[9] A. Skeppstedt, L. Ljung, and M. Millnert. Construction of composite models from observed data. *Int. j. Control*, 55(1):141–152, 1992.

[10] T. Söderström and P. Stoica. *System Identification*. Prentice-Hall Int., London, 1989.

[11] C.J. Stone. Consistent non-parametric regression (with discussion). *Ann. Statist.*, 5:595–645, 1977.

[12] J.E. Strömberg, F. Gustafsson, and L. Ljung. Trees as black-box model structures for dynamical systems. In *Proc. 1st European Control conference (ECC'91)*, pages 1175–1180, Grenoble, France, 1991.

[13] T.M.Cover and P.E. Hart. Nearest neigbor pattern classification. *Trans. IEEE Info. Theory*, IT-13:21–27, 1967.

On the Satisfiability of Symmetrical Constrained Satisfaction Problems

Jean-Francois Puget

ILOG SA, 2 avenue Gallieni, BP 85, F-94253 Gentilly Cedex, FRANCE
email : puget@ilog.fr

Abstract. Constraint satisfaction problems (CSP) are a class of combinatorial problems that can be solved efficiently by combining consistency methods such as arc-consistency together with a backtracking search. However these techniques are not adapted to symmetrical CSP. In fact one can exhibit rather small CSP that cannot be solved with consistency techniques. The relevance of this symmetry problem to real world applications is very strong since it can prevent a CSP solver to solve even small instances of real world problems. This paper describes a general solution for this kind of problems. Both a theoretical study and experimental results using the constraint-based library PECOS are provided.

1 Introduction

Constraint satisfaction problems (CSP) are a class of combinatorial problems that can be solved efficiently by combining local consistency methods, such as arc-consistency [8], together with a backtracking search. The idea of consistency method is to prune the search by preventing variable instantiation that are not consistent with the constraints of the problem. Different methods for this kind of pruning have been studied, e.g., [8], [7], [9], [10], [4], [3]. Sections 2 provides notation and semantics for this kind of problems.

However arc-consistency is not adapted to symmetrical CSPs. Avoiding symmetries is very important in real world applications since they can prevent a CSP solver from solving even small instances of real world problems. The work presented in this paper has been done while implementing a constraint-based programming library called PECOS [5]. This library has been used in a variety of industrial problems. In some of these problems, the success was only possible because we could avoid the combinatorial explosion caused by symmetries.

In fact one can exhibit rather small CSP that cannot be solved with consistency techniques. A simple problem is the pigeonhole CSP : one have to put $N + 1$ pigeons in N holes such that each pigeon is in a different hole. The problem can be viewed as a CSP by associating one variable to each pigeon, the value of which being the hole where the pigeon is placed. These variables must be different from each other. This CSP has no solutions since there is one more pigeon than holes. However even for small ($N = 20$) instances, this

problem cannot be solved by usual CSP solvers based on consistency techniques.

A typical CSP solver may proceed as follows. It first instantiates the first $N-1$ variables, thus using $N-1$ holes. Then, it tries to instantiate the last two but it fails since there is only one remaining hole for two variables. Thus the solver will try another instantiation for the first $N-1$ variables, and then fail. The solver will have to try every possible instantiation of the first $N-1$ variables to prove that there is no solutions. There are roughly $factorial(N-1)$ such possible instantiations. For $N=20$ this gives about 10^{17} different possibilities, which is clearly too much for current algorithms.

The problem is due to the symmetries of the CSP : any permutation of the variables of the CSP does not change the CSP. Section 3 formalizes the notion of symmetrical problems. A smarter approach would avoid the testing of permutations of a subproblem solution if it leads to a failure. This approach has been studied in [1] and [13] (see the related work section). We present another possible approach : always avoid the test of permutations of a subproblem solution, by adding new constraints, as explained in section 4. We then show how our method can solve a very difficult problem known as Ramsey's problem in section 5. We end the paper with a comparison with related work and some benchmark results.

2 Semantics

Two types of semantics have been proposed for constraint satisfaction problems. The first one describes a CSP as a graph, where nodes are variables, and arcs are binary constraints [8], [9], hence the name *arc-consistency*. The other, called Constraint Logic Programming describes constraint satisfaction as an extension of unification [6], [14]. We will describe an intermediate semantics : it does not use unification or Herbrand terms, and it makes a distinction between a constraint, and a constraint stated on specific variables (a literal). This semantics has been first proposed in [12].

In the remainder of the paper, $\mathcal{V}$ represents a set of variables, and $\mathcal{D}$ a set of constants. Intuitively, $\mathcal{D}$ is the set of possible values for the variables of $\mathcal{V}$. Consistency techniques associate to each variable the set of possibles values for that variable, usually called the domain of the variable.

Definition. (domains and values) *A* domain assignment *is a mapping from $\mathcal{V}$ to the powerset of $\mathcal{D}$. If v is a variable and dom a domain mapping, $dom(v)$ is called the* domain *associated to v by dom. A domain assignment dom is represented by the set of pairs $v/dom(v)$ for all v such that $dom(v) \neq \mathcal{D}$. A variable is* instantiated *by a domain assignment dom, iff $dom(v)$ is a singleton $\{d\}$. In that case, d is called the* value *of the variable.*

The semantics of a constraint is usually defined be the set of tuples satisfying the constraint. A literal is a constraint applied to variables:

Definition. (constraint, literal) *A* constraint *C of arity n is defined by a subset* ext(C) *of the cartesian product $\mathcal{D}^n$. A constrained literal is a formula $C(v_1, \ldots, v_n)$, where C is a constraint of arity n, and $v_1, \ldots, v_n$ are variables of $\mathcal{V}$.*

A problem is then defined by a set of variables, a domain and a set of constrained literals:

Definition (CSP) *A* Constraint Satisfaction Problem *is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{C}$ is a conjunction of constrained literals.*

The semantics of a constrained problem is the following:

Definition (solutions) *An* interpretation *is a domain assignment that instantiates all the variables of $\mathcal{V}$, i.e., $\forall v \in \mathcal{V}$, $\exists d \in \mathcal{D}$, $dom(v) = \{d\}$.*
A constrained literal $C(v_1, \ldots, v_n)$ is satisfied *by an interpretation dom iff*
$(dom(v_1), \ldots, dom(v_n)) \in ext(C)$.
An interpretation dom is a solution *for a problem $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ iff every constrained literal in $\mathcal{C}$ is satisfied by dom.*

For instance, if
$\mathcal{V} = \{x, y\}$,
$\mathcal{D} = \{0, 1, 2\}$,
$C = x \neq y \wedge y \neq z \wedge z \neq x$
then a possible domain assignment is
$x/\{0, 1, 2\}, y/\{1, 2\} z/\{1, 2\}$

There are six solutions for this problem, including:
$x/\{0\}, y/\{1\}, z/\{2\}$
$x/\{0\}, y/\{2\}, z/\{1\}$.

The usual way to solve CSP problems is to use an enumeration technique together with a local consistency prunning, such as *arc-consistency* [8]. Several local consistency algorithms have been proposed, including AC-3 [8], AC-4 [9], GAC-4 [10], AC-5 [3].

In the remainder of the paper, we assume that $\mathcal{D}$ is a set of integers, together with the usual ordering. We will use four basic constraints: $=$, $\neq$, $\leq$ and $<$. The above semantics can be extended when there is an ordering relation on the set of values $\mathcal{D}$ [2]. This semantics allows for a very simple definition of the constraints $=$, $\neq$, $\leq$ and $<$. However these constraints can be defined using the non-ordered semantics presented here. For instance, if $\mathcal{D}$ is $\{0, 1, 2\}$, the semantics of the con-

straints is the following:

$ext(=) = \{(0,0),(1,1),(2,2)\}$
$ext(\neq) = \{(0,1),(1,0),(0,2),(2,0),(1,2),(2,1)\}$
$ext(\leq) = \{(0,0),(0,1),(0,2),(1,1),(1,2),(2,2)\}$
$ext(<) = \{(0,1),(0,2),(1,2)\}$

3 Symmetrical CSPs

A symmetrical problem is a problem where some permutations of the variables map a solution onto another solution. We first define more precisely what is a permutation of the variables of a problem.

Definition (permutation of variables) *A* permutation of the variables *of a CSP* $(\mathcal{V},\mathcal{D},\mathcal{C})$ *is a bijection from* $\mathcal{V}$ *to* $\mathcal{V}$. *If* τ *is a permutation of the variables of a CSP* $\mathcal{P} = (\mathcal{V},\mathcal{D},\mathcal{C})$ *the* permuted CSP $\tau(\mathcal{P})$ *is the CSP* $(\mathcal{V},\mathcal{D},\tau(\mathcal{C}))$ *defined by:*
$\tau(\mathcal{C}) = \{C(\tau(v_1),\tau(v_2),\ldots,\tau(v_n))$ *such that* $C(v_1,v_2,\ldots,v_n) \in \mathcal{C}\}$

We will speak of a permutation, instead of a permutation of variables, wherever it is not ambiguous to do so.

The notion of symmetrical constraint is the usual mathematical notion of symmetrical relation:

Definition (symmetrical constraint) *A constraint of arity* n *is symmetrical if for any permutation* τ *of the integers* $\{1,\ldots,n\}$,
$\forall a_1,\ldots,a_n \in \mathcal{D}, (a_1,\ldots,a_n) \in ext(C) \rightarrow (a_{\tau(1)},\ldots,a_{\tau(n)}) \in ext(C)$

For instance the $\neq$ constraint is symmetrical. This notion is the basis for an equivalence of problems with respect to symmetries:

Definition (equality with respect to symmetries) *Given a constraint* C, *two literals* C_1 *and* C_2 *obtained from* C *are* equal w.r.t. symmetries, *noted* $C_1 =_{sym} C_2$, *iff either*
$C_1 = C_2$
C_1 *and* C_2 *bear on the same variables, and* C *is symmetrical*

We are interested in the permutations that map a solution in a solution:

Definition (consistent permutation of variables) *A permutation* τ *is consistent with a CSP* (V, D, C) *if* $\tau(\mathcal{C}) =_{sym} \mathcal{C}$.

This consistency notion has good properties: any combination of consistent permutations is a consistent permutation, and the inverse of a consistent per-

mutation is also consistent. Thus we obtain the following result :

Theorem 1 (group of consistent permutations) *The set of the consistent permutations of the variables of a CSP with the composition law is a group.*

Definition (symmetrical CSP) *A* symmetrical CSP *is a CSP with at least one consistent permutation other than the identity permutation.*

The simplest example of a symmetrical CSP is the permutation CSP :
$\mathcal{V} = \{v_i\}_{0 \leq i < N}$
$\mathcal{D} = \{0, 1, \ldots, N-1\}$
$\mathcal{C} = (\forall i, j, 0 \leq i < j < N, v_i \neq v_j)$

If $N = 3$ the problem is :
$v_0, v_1, v_2 \in \{0, 1, 2\}$
$v_0 \neq v_1 \wedge v_1 \neq v_2 \wedge v_2 \neq v_0$

This CSP is symmetrical. Indeed any swap of two variables gives an equivalent CSP. For instance, on the 3 variable CSP, if we swap v_0 and v_1 we obtain the following constraints :
$v_1 \neq v_0 \wedge v_0 \neq v_2 \wedge v_2 \neq v_1$
which are the same as the original ones w.r.t symmetries.

More generally, consider the transposition of two variables v_i and v_j, defined as follows :
$\tau_{i,j} \equiv v_i \rightarrow v_j$
$v_j \rightarrow v_i$
$v_k \rightarrow v_k if k \neq i, k \neq j$

Such a permutation is consistent with the CSP, since it does not change the set of constraints. But, a classical algebraic result states that any permutation is a combination of such transpositions. Thus by theorem 1, any permutation of the variables of this CSP is consistent.

When a CSP is symmetrical, any consistent permutation of the variables define a permutation on the solutions of the CSP.

4 Symmetrical CSPs without solutions

This section presents the fundamental result at the core of our method, and then illustrates it on a simple example.

4.1 Valid reduction of a CSP

A simple way to remove symmetrical solutions is to add constraints to P in order to remove permutations of solutions. This can be formalised as follows :

Definition (reduction) *A CSP (V, D, C') is a* reduction *of the CSP (V, D, C) if* $\mathcal{C} \subset \mathcal{C}'$

We call this a reduction because of the following result :

Proposition *If P is a CSP and P' a reduction of P, then the set of solutions of P' is included in the set of solutions of P.*

When considering symmetries, we look for reductions that do not remove a whole class of solutions, i.e. such that for any solution σ of the CSP, there exists a solution of the reduced CSP which is a permutation of σ :

Definition (valid reduction) *P' is a* valid reduction *of CSP P if:*
P' is a reduction *of P*
For any solution σ of P there exists a permutations τ_σ of the variables of P consistent with P such that σ is a solution of $\tau_\sigma(\mathcal{P}')$

Let us first consider the 3 variables permutation problem :
$v_0, v_1, v_2 \in \{0, 1, 2\}$
$v_0 \neq v_1 \wedge v_1 \neq v_2 \wedge v_2 \neq v_0$

A valid reduction of this problem is obtained by adding the following constraints :
$v_0 < v_1 < v_2$

This is indeed a valid reduction. For instance, consider a solution of the CSP :
$\sigma = (v_0/2, v_1/0, v_2/1)$.
This solution is not a solution of the reduced CSP. However, consider the permutation :
$\tau_\sigma \equiv v_2 \rightarrow v_0$
$v_0 \rightarrow v_1$
$v_1 \rightarrow v_2$

The permuted reduced CSP has the following constraints :
$v_1 \neq v_2 \wedge v_2 \neq v_0 \wedge v_0 \neq v_1$
$v_1 < v_2 < v_0$

Now, σ is a solution of this permuted reduced CSP ! By finding such a permutation for each solution of the CSP, we can prove that the reduced CSP is a valid reduction. Let us do this for the general case. The permutation CSP is :
$\mathcal{V} = \{v_i\}_{0 \leq i < N}$
$\mathcal{D} = \{0, 1, \ldots, N-1\}$
$\mathcal{C} = (\forall i, j, 0 \leq i < j < N, v_i \neq v_j)$

We add the following set of constraints :

$\mathcal{C} = (\forall i, j, 0 \leq i < j < N, v_i \leq v_j)$

The reduced CSP is equivalent to :
$\mathcal{V} = \{v_i\}_{0 \leq i < N}$
$\mathcal{D} = \{0, 1, \ldots, N-1\}$
$\mathcal{C} = (\forall i, j, 0 \leq i < j < N, v_i < v_j)$

Proposition *This CSP is a valid reduction of the permutation CSP.*

The proof follows the idea used for the 3 variable example. If σ is a solution of this reduced permutation CSP, consider the following permutation :

$\tau_\sigma \equiv v_{\sigma(v_i)} \rightarrow v_i$

σ is then a solution of the permuted problem.

We have just proved that for any solution σ there exists a permutation of the variables such that σ is a solution of the permutation CSP. Thus the reduced CSP is a valid reduction.

4.2 Central result

The notion of valid reduction is very useful because of the following results :

Theorem 2 (sufficient condition for unsatisfiability) *A CSP has no solution if there exists a valid reduction of this CSP which has no solution.*

Proof : consider a CSP $\mathcal{P}$ and a valid reduction of it $\mathcal{P}'$ having no solutions. If σ is a solution of $\mathcal{P}$, then $\tau_\sigma(\sigma)$ is a solution of $\mathcal{P}'$, which contradicts our hypothesis.

Theorem 3 (A symmetrical problem has a valid reduction) *A symmetrical CSP has always a valid reduction*

Proof : Consider a symmetrical CSP $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$. Then there exists at least one permutation τ consistent with this CSP. Thus there exists a variable $v_i \in \mathcal{V}$ such that $\tau(v_i) = v_j \neq v_i$. Then one obtains a valid reduction $\mathcal{P}'$ of $\mathcal{P}$ by adding the constraint $v_i \leq v_j$. Indeed, if σ is a solution of $\mathcal{P}$, such that $\sigma(v_i) > \sigma(v_j)$, then σ is a solution of $\tau(\mathcal{P}')$.

These results are at the core of our method : the ideal goal is to be able to compute a non symmetrical valid reduction of any symmetrical problem by adding ordering constraints. If the reduction is not satisfiable, then the original problem is not satisfiable either. However the unsatisfiability proof is easier on the reduced problem since it has a smaller number of potential solutions.

4.3 The pigeonhole CSP

We first show how the preceding results can be used on a simple problem : the pigeonhole problem. This problem can be viewed as a CSP by associating one variable to each pigeon, the value of which being the hole where the pigeon is placed. The formal description of the problem is then similar to the one presented in previous section with one additional variable :

$\mathcal{V} = \{v_i\}_{0 \leq i \leq N}$
$\mathcal{D} = \{0, 1, \ldots, N-1\}$
$\mathcal{C} = (\forall i, j, 0 \leq i < j \leq N, v_i \neq v_j)$

The pigeonhole CSP has obviously no solution. It is also a symmetrical CSP as can be shown by the same considerations that was used for the permutation CSP. The valid reduction is obtained by adding the same ordering constraints :

$\mathcal{V} = \{v_i\}_{0 \leq i \leq N}$
$\mathcal{D} = \{0, 1, \ldots, N-1\}$
$\mathcal{C} = (\forall i, j, 0 \leq i < j \leq N, v_i < v_j)$

Proposition *This CSP is a valid reduction of the pigeonhole CSP.*

The proof is similar to the one for the permutation CSP.

The reduced pigeonhole CSP is easily shown to be unsolvable, since local consistency is sufficient to detect the impossibility. This proves that the original CSP is also unsatisfiable using theorem 2.

5 The Ramsey problem

We will now consider a much more difficult problem which has been first solved using our technique see [11], namely the Ramsey problem. Consider the complete graph with n nodes (each node is connected to every other node). The problem is to color the edges of this graph with 3 colors, such that for any 3 nodes $n1, n2, n3$, the three arcs $(n1, n2)(n2, n3)(n3, n1)$ do not have all three the same color (but two of them can have the same color). For $N = 16$, this problem has a lot of solutions. For $N = 17$ there is no solution. It is not very difficult to write a program for finding solutions for $N \leq 16$. The challenge is to write a program which solves the problem for $N \leq 16$, and proves that there is no solution for $N = 17$.

5.1 The cardinality constraint

In the previous problem we have considered simple orderings bearing on the values taken by the variables. We will now use another kind of ordering which may be very useful for removing symmetries. These orderings uses the *cardinality* constraint defined in [11]. This constraint is defined as follows.

If $\{x_0, x_1, \ldots, x_n\}$ is a finite subset of $\mathcal{V}$, and num is an integer variable :
$num = count(d, \{x_0, x_1, \ldots, x_n\})$
states that the number of variables in $\{x_0, x_1, \ldots, x_n\}$ taking the value d is equal to the value of the variable num.

proposition *The* count *constraint is symmetrical.*

This constraint is less general than Van Hentenryck's [15] cardinality operator, but is more efficient as shown in [11].

5.2 The symmetrical CSP

The Ramsey problem can be encoded with one variable $e_{i,j}$ representing the color of the edge from node i to node j The CSP is thus :
$\{e_{i,j}\}_{0 \leq i,j \leq N-1} \in \{0, 1, 2\}$
$\forall i, j, k, i \neq j \neq k e_{i,j} \neq e_{i,k} \vee e_{i,k} \neq e_{k,i} \vee e_{k,i} \neq e_{i,j}$
$\forall i, j, e_{i,j} = e_{j,i}$

This representation is symmetrical since the count constraint is symmetrical : two nodes can be swapped. The corresponding permutations are the followings :
$\rho_{i,j} \equiv e_{i,k} \rightarrow e_{j,k}$
$e_{j,k} \rightarrow e_{i,k}$
$e_{k,l} \rightarrow e_{k,l} if f k, l \notin \{i, j\}$

By theorem 1, any permutation of the nodes yields a consistent variable permutation.

5.3 The valid reduction of the CSP

In order to reduce this problem using ordering constraints, one must find quantities that are changed by one of these swaps. One such quantity is the number of edges going from one node i having a given color $color$. Let us call this $numcolor(i, color)$. By swapping nodes, one can obtain a solution where the number of edges with color 0 is the biggest among the $numcolor(i, color)$ Moreover, one can suppose that the values of $e_{i,1}$ are increasing.

The reduced CSP is then
$\{e_{i,j}\}_{0 \leq i,j \leq N-1} \in \{0, 1, 2\}$
$\forall i, j, k, i \neq j \neq k e_{i,j} \neq e_{i,k} \vee e_{i,k} \neq e_{k,i} \vee e_{k,i} \neq e_{i,j}$
$\forall i, j, e_{i,j} = e_{j,i}$
$\forall i, colornumcolor(i, color) = count(color, \{e_{i,0}, e_{i,1}, \ldots, e_{i,N-1}\})$
$\forall i, colornumcolor(0, 0) \geq numcolor(i, 0)$
$e_{0,1} \leq e_{0,2} \leq \cdots \leq e_{0,N-2} \leq e_{0,N-1}$

Proposition *This CSP is a valid reduction of the original Ramsey CSP*

Proof. If σ is a solution of the Ramsey CSP, let i_0 be the rank of the node with the biggest number of edges of colors 0. Then by definition, we have that :
$\forall i, color,\ numcolor(i_0, 0) \geq numcolor(i, color)$
Then we can permute the other nodes so that edges starting from node i_0 have increasing colors. The fact that any node permutation yields a consistent permutation of the variables, terminates the proof.

This reduced CSP still has some symmetries. However the number of symmetries is sufficiently small for the problem to be solved in a reasonable amount of time.

6 Concluding remarks

6.1 Related work

The starting idea of our method is to remove symmetrical solutions by changing the CSP to be solved. Other recent work concentrates on a different idea : how can we design a smarter CSP solver that would avoid the testing of permutations of a subproblem solution if it leads to a failure. This approach has been studied independently in [1] and [13]. In both approaches the solver analyses the CSP when a failure happens, and tries to recognize symmetries of the CSP related to the failure. In other words, they are looking for symmetries that leave the cause of the failure unchanged. Then the solver avoids testing symmetrical domain assignments.

The main difficulty with these approaches is the recognition of symmetries. Solving this recognition problem in its most general form is at least as difficult as proving the unsatisfiability of a symmetric CSP. Thus both papers propose an approximate solution : good heuristics that search for most commonly encountered symmetries are used. Using these approaches, the Ramsey 17 problem has been solved independantly from us.

We think that our approach is more efficient, although less automated, because we reduce the total number of potential solutions before searching a solution. The next section provides evidence for this claim.

6.2 Results

This work has been done during the implementation of a constrained programming library described in [5] called PECOS. This library provides support for constrained variables, domains, and constraints as described in this paper. It is available either as a Lisp library or as a C++ library

Another CSP is used as a benchmark for symmetries : The Schur problem. One has to place a given number N of balls in 3 boxes. The balls are numbered

from 1 to 13. The ball number i and the ball numbered $2i$ cannot be in the same box. The balls numbered i j and $i+j$ cannot be all three in the same box. With $N = 13$ there is a solution, and there is no solution for $N = 14$.

All the CSP discussed in this paper have been implemented using PECOS. We give cpu time needed to solve with the C++ version on a sparcstation 2 these CSPs. Times for the other two approaches are taken from [13] and [1] who both use a sparc workstation. Thus times can be compared. They are given in seconds.

	PECOS	[13]	[1]
ramsey 14 (one solution)	4.07	4	2.82
ramsey 15 (one solution)	2.57	42	50.73
ramsey 16 (one solution)	8.8	50.76	85
ramsey 17 (proof of unsatisfiability)	1.51	??	30
schur 13 (one solution)	0.02	0.16	0.05
schur 14 (proof of unsatisfiability)	0.14	0.86	1.33

6.3 Summary and future work

We have studied a method for removing symmetries in CSP. This method relies on a theoretical analysis of the problem, based on a study of the consistent permutations of the variables of a problem. We then presented a general method, based on the notion of valid reductions of CSP. This technique can be used to solve hard CSP such as the Ramsey CSP.

However our technique is not fully automated since a valid reduction has to be chosen. We think that the best approach to study would be a combination of our technique together with the one described in [13] and [1]:

- the first stage would be the recognition of symmetries of the CSP by analysing the constraints of the CSP. This step would benefit from the heuristics used in [13] and [1].
- The second stage would be an automatic generation of additional ordering constraints yielding valid reductions of the CSP.

Theorem 3 suggests that the second step is not totally a dream.

References

1. Benhamou B., and Sais L., "Theoretical study of symmetries in propositional calculs and applications", *in proceedings of CADE 92* june 92.
2. Casean Y, Puget JF, "Constraints on Order Sorted Domains", submitted.
3. Deville, Y., and Van Hentenryck, P, "An Efficient Arc Consistency Algorithm for a Class of CSP Problems", *in proceedings of IJCAI 1991, pp 325-330.*
4. Gusgen, Hertzberg "Some fundamental properties of local propagation methods" *Art. Int. pp 237-247, 1988.*

5. Ilog, *PECOS 1.2 reference manual, december 92 1992*
6. Jaffar J., and Lassez J.-L. Constraint Logic Programming, in proceedings of POPL 1987, Munich, January 87.
7. Lauriere J.L., A language and a Program for Stating and Solving Combinatorial Problems, *Art. Int. 10(1)*.
8. Mackworth A.K., "Consistency in networks of relations", *Art. Int 8, pp 99-118, 1977*
9. Mohr, Henderson "Arc and path consistency revisited" *Art. Int. 28, pp 128-233, 1986.*
10. Mohr, Masini, "Good Old Discrete Relaxation", *in proceedings of ECAI 1988.*
11. Puget J.-F., "Pecos : a High Level Constraint programming Language" *in proceedings of SPICIS 92*, Singapore, Sept 92.
12. Puget J.-F., "Programmation par contraintes oriente'e objet" *in proceedings of Tenth international conference on expert systems and applications*, Avignon, June 92 (in French).
13. San Miguel Aguire, "HOW to use symmetries in Boolean constraint solving" in A. Colmerauer and F. Benhamou, editors, *Selected Papers on Constraint Logic Programming* (to appear) MIT Press.
14. Van Hentenryck, P., *Constraints Satisfaction in Logic Programming*, MIT press, 1989.
15. Van Hentenryck, P., Deville Y., "The cardinal operator : A new logical connective for constraint logic programming" *in proceedings of ICLP 91, pp 745- 759.*

A Logical Reconstruction of Constraint Relaxation Hierarchies in Logic Programming

Allen L. Brown, Jr.
Surya Mantha
Toshiro Wakayama

Webster Research Center, Xerox Corporation

Abstract. *We propose an extension to Definite Horn Clauses by placing partial orders on the bodies of clauses. Such clauses are called* relaxable clauses. *These partial orders are interpreted as a specification of* relaxation criteria *in the proof of the consequent of a relaxable clause,* i.e., *the order in which to relax the conditions of truthhood of the consequent if all the goals in the body cannot be satisfied. We present a modal logic of preference that enables us to characterize these preference orders, both syntactically and semantically. The richer structure of the modal preference models reflects these preference orders; something that is absent in the essentially flat structure of traditional Herbrand models. A variant of SLD-resolution that generates solutions in the preferred order is presented. The notion of* control as preference *is introduced as a first step towards specifying control information in a logically coherent fashion. Relaxable Horn clauses can be used to succinctly specify constraint problems in formal design. It is worth noting that the development of preference logic was driven by the desire to characterize declaratively, problems in document layout. In [4] we give a completely declarative account of the stable models of a general logic program. The reader is referred to [3], [5]and [14] for a detailed account of nonmonotonicity as preferential reasoning, the soundness and completeness proofs for the logics and applications to Artificial Intelligence, such as deontic reasoning.*

1 Motivations for Relaxable Specifications

One of the many attractions of logic programming is that it is an excellent executable specification language. Corresponding model-theoretic, fixed-point and operational semantics (via SLD-resolution) ensure that an atomic fact is in the denotation of a logic program if and only if it is in the success set of the program. The least Herbrand model of a logic program fully specifies the *state of affairs*, so to speak. Insofar as the purpose of a formal specification is to spell out conditions under which certain statements are true and also those statements that are actually true, the language of Horn clause logic is sufficient (clauses and atomic facts do the respective jobs). At its core, however, a logic programming interpreter is a theorem prover, and this is manifested in the *binary* nature of the query mechanism. Given an atom as a query q and a specification (or program) P, the system returns *true* if q is a logical consequence of P and either returns

false or loops otherwise. A desirable feature of a formal specification system is that it should be possible to *relax* the requirements on the truth of a statement in a disciplined and semantically coherent way. The relaxation regime should be specifiable in a totally declarative fashion.

Before proceeding any further, we must formalize the notion of a constraint. For our purposes it is sufficient to view a constraint as a relationship among objects, or in other words, a predicate. We shall take a constraint to be a declarative description of a relation along with *methods* to satisfy (or enforce) the relationship. These methods could be specified by procedures or declaratively, using Horn clauses. We shall work with Horn clauses. Consider the example in

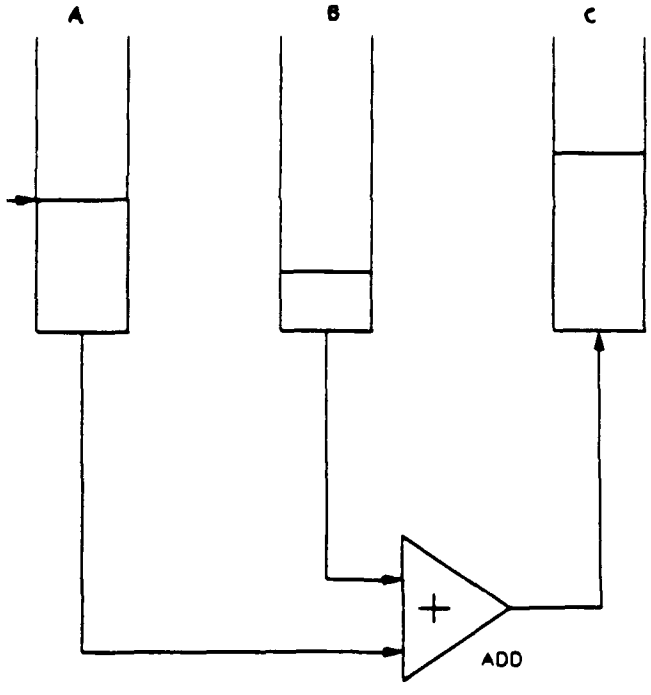


Fig. 1. A Relaxable Constraint

Figure 4.1. The system has four hard constraints: $A + B = C$, $A \geq 0$, $B \geq 0$, and $C \geq 0$. The first constraint expresses the fact that the amount of a certain fluid F in container C is a sum of the amounts in A and B. The others state that the minimum amount of F that each container can hold is bounded below by 0. Let the overall constraint expressed by this example be $ADD(A, B, C)$, and the rule

$$ADD(A, B, C) :- A + B = C, A \geq 0, B \geq 0. C \geq 0.$$

Suppose Figure 1 actually appears on the computer screen with the cursor at A. If we drag the cursor up – the semantics of this action being that more fluid is poured into A – how should the system react in order to satisfy the constraint ADD. As it stands, the system is underconstrained. It could either reduce the amount of fluid in B or increase the amount in C. We would prefer that C go up as A goes up (represented by the constraint $INCREASING(A, C)$, whose definition we shall not worry about here). Suppose, we have a preference as to the amount of fluid that container C should ideally hold, say less than or equal to 10 units. These two constraints are not hard constraints, in the sense that they need not be satisfied under all circumstances. However, we prefer $C \leq 10$ to $INCREASING(A, C)$ because of reservations about the strength of the container C (for instance!). Assume that the system starts in the initial state (6, 2, 8) i.e., A has 6 units, B 2 and C 8. By increasing A by 1, the resulting state of the system is described by the vector (7, 2, 9). The system was able to satisfy all the six constraints on it. On further increasing A, we get the following trace: (8, 2, 10), (9, 1, 10), (10, 0, 10), (11, 0, 11). In the change from the vector (8, 2, 10) to (9, 1, 10), the system was not able to satisfy all the constraints. Because $C \leq$, 10 was preferred to $INCREASING(A, C)$, the latter was sacrificed. The last vector shows a state where $C \leq 10$ could not be satisfied. The fact that $INCREASING(A, C)$ is now resatisfied is incidental.

How does one represent such knowledge? A rather attractive way that suggests itself is the following.

$ADD(A, B, C) \leftarrow$
required: $A + B = C$,
required: $A \geq 0$,
required: $B \geq 0$,
required: $C \geq 0$,
prefer: $C \leq 10$,
default: $INCREASING(A, C)$.

This appealing syntax where the goals in the body are *annotated* with their respective *strengths* has been proposed in the literature [2]. Such formalisms are, however, ad hoc because they do not give a declarative, model-theoretic semantics for such clauses. The logical status of such annotations is left unspecified. It should be noted that these proposals are an extension of the CLP framework and they attach annotations only to constraints, and not to all the goals in the bodies of clauses. However, viewed as formulae in logic, they do not *denote*. We propose a more general mechanism whereby preference orders are imposed on all the literals in the body of a clause.

2 Relaxation Hierarchies in Logic Programming

Our stated goal is to devise a language of relaxable specifications in which the relaxation criteria have due status in the model theory. Drawing inspiration

from the annotated (with *strengths* such as **required**) concrete syntax presented above, we define a new category of clauses called *relaxable clauses.*

Definition 1. A *relaxable clause* is a triple $\langle H, B, P\rangle$, where H is an atom, B is a set of atoms and P is a partial order on B.

Thus, an ordinary clause is a relaxable clause with a an empty partial order.

Definition 2. A *relaxable logic program* is a finite set of *relaxable clauses.*

Definition 3. If P is a partial order over a set S, then an element x of S is maximal in P iff there is no element y in S such that $x < y$.

Definition 4. Let $C = \langle H, B, P\rangle$ be a relaxable clause. The *minimal support set* of H in the context of C is $max(P)$, where $max(P)$ is the set of maximal elements in the partial order P.

What is the informal reading of a relaxable clause? An atomic query Q can be proved using a relaxable clause $C = \langle H, B, P\rangle$ of a relaxable program R, if Q unifies with H with θ being the mgu, and at least the maximal elements of P, *i.e* $\theta(max(P))$ can be proved using R. Ideally, we would like all of the goals in t' body B to be provable. If that is not possible, the partial order P specifies the order in which the body can be relaxed. The satisfaction of a goal is of higher priority than of all the goals that are lower than it in the partial order. Of course, the same information can be used for goal selection by the SLD interpreter, but more on that later. Consider the following (relaxable) logic program GP_1 (for *goal preference*) with four clauses.

$\langle p(X), \{q, r(X), s(X)\}, \{s(X) < q, s(X) < r(X), r(X) < q\}\rangle$
$\langle q, -, -\rangle.\ \langle s(2), -, -\rangle.\ \langle r(1), -, -\rangle.$

What is the behavior of the above program given the goal $\leftarrow p(Y)$? Since the goal unifies with the head of the first clause (and the only clause for p), we would, ideally, like to satisfy all the goals in the body. However, the satisfaction of the maximal elements – $\{q\}$ in this case – is sufficient. Thus, both $p(1)$ and $p(2)$ are provable. The relaxation criteria, however, put a preference order over the two solutions to this query. Because $r(X)$ is higher than $s(X)$ in the partial order, the solution $p(1)$ will be preferred to the solution $p(2)$. Can a similar behavior be achieved using definite Horn clauses? A first approximation is the following logic program CP_1 (for *clause preference*).

$p(X) \leftarrow q, r(X), s(X).$ $p(X) \leftarrow q, r(X).$ $p(X) \leftarrow q, s(X).$
$p(X) \leftarrow q.$ $q.$ $s(2).$
$r(1).$

The information about the partial order in the first clause of GP_1 is captured by the *textual order* of the p-clauses in CP_1. We shall call this the *clause preference translation.*

Definition 5. The clause preference translation of a relaxable program GP_1 is the union of the ordered sets obtained by the clause preference translation of each relaxable clause in GP_1.

We have a total order in our example. A partial order is more cumbersome to represent using the *try order* of clauses. Moreover, this information has no status in the least Herbrand model of CP_1 (if CP_1 is considered to be an ordinary, unordered collection of clauses) $-\{p(1), p(2), q, s(2), r(1)\}$ - which is a flat structure. The fact that $p(1)$ is preferred over $p(2)$ is not represented in the semantics. It is solely a consequence of the textual order of the clauses. It is this added *intensionality* that we would like to capture explicitly in a logic.

Another possible translation of the relaxable logic program GP_1 is into PP_1 (for *program preference*), a set of four logic programs $\{\pi_i | 1 \leq i \leq 4\}$. The set of clauses $\{q, s(2), r(1)\}$ is common to all the π_i. They differ in the definition of the predicate p, and π_1 has the first p-clause of CP_1, π_2 the second and so on. We now have four competing logic programs to choose from when solving a query. What is required to complete the translation is a fifth component that places a preference order among these programs - an *arbiter*, so to speak. The framework of logic programming is not rich enough to enable us to express such preferences and thus complete the translation into PP_1. The reader will have noticed that in the course of these translations what has been revealed is that relaxable specifications can be looked upon as either *goal preference*, *clause preference* or *program preference* logic programs, where the partial order is on the goals, clauses and programs respectively.

3 Preference as a Modality

The notion of *preference* is fundamental to computing. As should be obvious, the notion of *minimality* derives from it, and the search for minimality is a recurring theme in artificial intelligence and theoretical computer science. It has been studied quitely extensively in such diverse areas as decision theory, operations research, economics, ethics, and philosophical logic [11] [8], [15], [16], [7], [9].

As a first approximation, a logic of preference should concern itself with the study of preference principles acceptable upon abstract and formal grounds rather than upon any particular theory of preferability determination. Competing - and mutually inconsistent - theories of preference have been proposed by economists, mathematicians and philosophical logicians among others. The work in economics and utility theory [11], [13] relates qualitative notions of preference to quantitative notions of probability and desirability, but places rather stringent conditions on the preference relation, i.e., that it be an irreflexive, asymmetric and transitive relation. In fact, much of the controversy and debate in this whole area has been around the question: *what are the logical properties of preference?* Much has been written for and against the *transitivity* of preference [10], [6]. Even such a seemingly innocuous property as *asymmetry* has come under criticism [1].

Elsewhere [5], [14] we have argued about what the right *granularity* of preference should be. In almost all work in philosophical logic, utility theory and economics, preference is taken to be a binary relation among propositions, i.e., those objects that can be represented by sentences of the underlying logical language. The scope of preference is thus, extemely *local*. Having committed to such

a fine level of granularity, one has to make rather strong commitments as to what *logical* properties the preference relation enjoys under all circumstances. The use of preference in nonmonotonicity has seen a drastic shift from the very local to the *very global*. Thus, in circumscription, we talk about truth in **all minimal models**. The inference rules of default logic and other nonmonotonic logics appeal to certain *global condtions* as well. Such a shift has had a disastrous impact on the computability of preference. It is ironic that this should have happened in computer science, of all disciplines.

Consider the preference proposition pPq. According to the usual reading, it means *p is preferred to q*. It can be claimed reasonably that p and q do not obtain in a vacuum after all. Thus, it means that *p is preferred to q independent of everything else, or that p is preferred to q given that everything else is the same, or some intermediate point between these two extremes.* In our view, the binary preference relations between propositions are secondary and can be derived from the preference orderings among possible worlds. The fundamental relation of preference is among possible worlds, i.e. those objects that are described by a possibly infinite collection of propositions. What is important to keep in mind is that we do not take individual sentences of the logical language to describe the *states of affairs* over which the preference relation ranges. This makes it difficult to characterize the preference relation using a binary relation P, because it is not convenient to explicitly talk about the referents of this relation. We need a syntactic means of characterizing these preference orders among possible worlds. In short, we seek a modal operator for preference that would play the role that $\Box$ does in alethic modal logic.

3.1 Logic of Feasible Preference $\mathcal{P}_1$

$\mathcal{P}_1$ is a modal logic of two relations that *interact* with each other. The motivation for this interaction is to capture the intuition that in order to get to the *optimal* (or best) world, one needs to be able to talk about worlds that are feasible from the standpoint of the current world. The motivation underlying this whole enterprise is to devise a formal language and logic in which optimization problems can be stated precisely. Thus, if w_2 is feasible from w_1, and $w_1 \preceq w_2$ and $w_2 \not\preceq w_1$, then it is possible to move from the solution w_1 to w_2. If, however, w_2 were not feasible from w_1, then it would not be possible to move from w_1 to w_2 even if w_2 were preferred to w_1. This interaction between the two relations is fundamental to modeling any situation that is of computational interest; in particular all search based computation. Drawing an analogy from classical mathematics, *feasibility* corresponds to *continuity* of the domain of computation (just as preference corresponds to maximization).

Syntax : We add to the language $\mathcal{L}_m$ of a normal modal logic - equipped with the modal operators $\Box$ and $\Diamond$ - a new unary modal operator P_f and its associated formation rule, i.e., if A is a formula then $P_f A$ is a formula.

Semantics : A $\mathcal{P}_1$ *preference frame* $\mathcal{M}$ is a triple of the form $(\mathcal{W}, \mathcal{R}, \preceq)$ where $\mathcal{W}$ and $\mathcal{R}$ are as in standard Kripke frames and $\preceq$ is a binary relation over

$\mathcal{W} \times \mathcal{W}$ which is a subset of $\mathcal{R}$. A $\mathcal{P}_1$ *preference model* is a $\mathcal{P}_1$ preference frame with a valuation function $\mathcal{V}$ that determines the truth of atomic formulae at individual worlds. Assuming the usual valuation of formulae at possible worlds, the semantics of the modal operators are given by:

- $\models^w_{\mathcal{M}} \Box F$ iff $\forall v \in \mathcal{W}\ w \mathcal{R} v \ \rightarrow \models^v_{\mathcal{M}} F$
- $\models^w_{\mathcal{M}} P_f F$ iff $\forall v \in \mathcal{W} \models^v_{\mathcal{M}} F \wedge w \mathcal{R} v \rightarrow w \preceq v$

Axiomatics : We assume that $\mathcal{P}_1$ is equipped with all the axiom schemes and rules of standard propositional logic, and the normal modal logic $\mathcal{K}$. In addition, we have the following axioms and rule.

PPS : $\vdash \Box \neg A \rightarrow P_f A$

PIR : $\vdash P_f A \rightarrow \neg A$

T : $\vdash \Box A \rightarrow A$

PI if $\vdash (A_1 \wedge \cdots \wedge A_n) \rightarrow A$ then $\vdash (P_f \neg A_1 \wedge \cdots \wedge P_f \neg A_n) \rightarrow P_f \neg A$ for $n \geq 0$

PIR is valid in the class of preference frames with an *irreflexive* preference relation. **T** is valid in the class of preference frames with a *reflexive* feasibility relation. **PIR** and **T** are needed to show the completeness of $\mathcal{P}_1$. Some axioms and rules that are valid are

PI* $P_f(A \rightarrow B) \rightarrow P_f B$. **PA** $(P_f A \wedge P_f B) \rightarrow P_f(A \wedge B)$.

PF $P_f \bot$. **PGN** $P_f A \rightarrow P_f(A \wedge B)$.

PST $P_f(A \vee B) \rightarrow P_f(A)$. **PEQ** if $\vdash (A \leftrightarrow B)$ then $\vdash (P_f A \leftrightarrow P_f B)$

The language of $\mathcal{P}_1$ is rich enough to allow us to express general preference principles. We refer the reader to [14], [5] for details. In the rest of this paper, we work with the first-order version of $\mathcal{P}_1$ with fixed domains across worlds, terms as rigid designators.

4 Horn Preferential Theories

Central to our enterprise will be the notion of a *preferential theory*. We first give some informal motivational remarks. Consider the typical structure of an optimization problem. The following components can be immediately identified: a set of constraints that have to be satisfied in all solutions, the space of solutions, and, a set of preference criteria that picks out one or more solutions from among the space of solutions as optimal solutions. The language of preference logic is expressive enough to enable us to specify all three components in a succinct fashion. A *preferential theory* is, informally, the statement of an optimization problem in the language of preference logic.

Definition 6. A *preference clause* is the universal closure of a formula of the form $\bigwedge_k M_k \rightarrow P_f(\bigwedge L_j)$ where L_j and M_k are general literals, i.e., literals (possibly) adorned with $\Box$ and $\Diamond$.

Preference clauses are sufficient for most purposes. The treatment of preferential theories, with iterated modalities is left to a later date.

Definition 7. An *arbiter* is a finite collection of preference clauses.

Definition 8. A *preferential theory* is the conjunction of a first-order theory T and an arbiter $\mathcal{A}$.

Definition 9. A *preinterpretation* for a language L is a set D (the domain of interpretation – we restrict ourselves to the single sorted case) and a mapping from terms (including variables) to D.

Definition 10. Given a preinterpretation I', an I'-based interpretation I is I' together with a mapping from (n-place) predicate symbols to subsets of D^n.

Definition 11. Given a preinterpretation I', an I'-based preferential structure is a structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \preceq, \mathcal{V} \rangle$ such that $\mathcal{V}$ assigns I'-based interpretations to members of $\mathcal{W}$.

In this paper, we shall be interested in a particular class of preferential theories called Horn preferential theories. Further, the only preinterpretation of interest to us will be the Herbrand preinterpretation, i.e., the free interpretation of terms.

Definition 12. Let $\{\pi_i | i \in I\}$ be a finite collection of definite Horn theories (standard definitions from [12]). Thus each π_i is a conjunction of definite Horn clauses. Let μ be a definite Horn theory as well. A *modular theory* T_m is defined to be:

$$\mu \wedge (\bigwedge_j \Diamond(\pi_j \wedge \neg(\bigvee_{k \neq j} \pi_k)))$$

where j and k range over I.

Intuitively, μ corresponds to the fixed part of a specification. Thus μ must be satisfied at all worlds in all models in which the modular theory is true. The various π_i are dispensable. Let us call them the *transients* and μ the *invariant*. The latter conjunct in the above formula for a modular theory imposes the following condition on the models of a modular theory.

Lemma 13. *If $\mathcal{M}$ is a preference model of a modular theory T_m (i.e., $\mathcal{M} \models T_m$), then*

- *For every world w in $\mathcal{M}$, and for every transient π_i in T, there exists some w-feasible world v, such that $\models^v_{\mathcal{M}} \pi_i$ and $\not\models^v_{\mathcal{M}} \pi_j$ for all $j \neq i$.*

Definition 14. A *solution* to a modular theory is given by $\mu \wedge \pi_s$ for any s in I.

We shall assume that $\pi_i \not\subseteq \pi_j$ for $i \neq j$. Also $\pi_i \not\subseteq \mu$ for any i. Of course, this is quite easy to arrange by introducing new dummy predicates that are unique to the respective programs.

Given a modular theory that has n transients (and thus n solutions), we would like to be able to specify preference criteria that determine optimal solutions. A set of preference criteria act as an *arbiter* in determining optimal solutions. Let us denote the arbiter by $\mathcal{A}$. We take the arbiter to be a finite set of preference clauses.

Definition 15. A Horn preferential theory is of the form

$$\mu \wedge (\bigwedge_j \Diamond(\pi_j \wedge \neg(\bigvee_{k \neq j} \pi_k))) \wedge \mathcal{A}$$

where $\mathcal{A}$ is an *arbiter* and the first part is a modular theory.

Definition 16. Let I' be a preinterpretation and let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two I'-based preferential structures. $\mathcal{M}_1 \leq \mathcal{M}_2$ iff

1. There exists ϕ, a one-to-one mapping from $\mathcal{W}_1$ to $\mathcal{W}_2$ such that $\forall w \in \mathcal{W}_1 \mathcal{V}_2(\phi(w)) \supseteq \mathcal{V}_1(w)$.
2. $\phi(\mathcal{R}_1) \supseteq (\mathcal{R}_2$ restricted to $\phi(\mathcal{W}_1 \times \mathcal{W}_1))$ (by a minor abuse of notation).
3. If in the above condition, equality holds then, $\preceq_2$ restricted to $\phi(\mathcal{W}_1 \times \mathcal{W}_1) \supseteq \phi(\preceq_1)$.

Lemma 17. Let T_P given by

$$\mu \wedge (\bigwedge_j \Diamond(\pi_j \wedge \neg(\bigvee_{k \neq j} \pi_k))) \wedge \mathcal{A}$$

be a Horn preferential theory. Let H be the Herbrand preinterpretation over the language of the preferential theory. There is a unique H-based $\leq$-minimal preferential structure $\mathcal{M}_H$ such that $\mathcal{M}_H \models T_P$. $\mathcal{M}_H = \langle \mathcal{W}_H, \mathcal{R}_H, \preceq_H, \mathcal{V}_H \rangle$ where

- *$\mathcal{W}_H = \{w_1, \cdots, w_n\}$, where n is the cardinality of the solution space,*
- *$\mathcal{R}_H = \mathcal{W}_H \times \mathcal{W}_H$,*
- *$\mathcal{V}_H(w_i) = M_{\mu \wedge \pi_i}$ where $M_{\mu \wedge \pi_i}$ is the least Herbrand model of $\mu \wedge \pi_i$.*
- *$\preceq_H$ is the smallest subrelation of $\mathcal{R}_H$ that satisfies the arbiter.*

The intended preference model of a Horn preferential theory is its least preference model.

5 Relaxable Logic Programs as Horn Preferential Theories

We now have the logical machinery to complete our *program preference* translation that we began in an earlier section. We shall use preference logics to specify the fifth component of our translation of relaxable logic programs. Consider the *program preference* translation of the first clause of our example relaxable program GP_1. We shall generalize this construction to a set of such relaxable clauses. Let GC_1 denote

$$\langle p(X),\ \{q,\ r(X),\ s(X)\},\ \{s(X) < q,\ s(X) < r(X),\ r(X) < q\} \rangle$$

This gives rise to the four clauses

$C_1 : p(X) \leftarrow q, r(X), s(X)$. $C_2 : p(X) \leftarrow q, r(X)$.

$C_3 : p(X) \leftarrow q, s(X)$. $C_4 : p(X) \leftarrow q$.

Let π_i be the program with the clause C_i. What is left to be specified is the arbiter that imposes the partial order among these clauses. The purpose of the arbiter is to perform *preferential maximization* in the context of the truth of the predicate $p(X)$ for all X. The arbiter for the first clause of GP_1 is

$p(X) \wedge r(X) \wedge \neg s(X) \rightarrow P_f(p(Y) \wedge r(Y) \wedge s(Y))$

$p(X) \wedge \neg r(X) \wedge \neg s(X) \rightarrow P_f(p(Y) \wedge s(Y))$

$p(X) \wedge \neg r(X) \rightarrow P_f(p(Y) \wedge r(Y))$

The arbiter clauses use the partial order on the body of the relaxable clause to specify which p worlds are better.

Definition 18. Let $GC = \langle H, B, PO \rangle$ be a relaxable clause. Let B_i be an element of B. $C(B_i, GC)$ is the *constant set* of B_i while proving H in the context of GC. It is given by the set: $\{B_j \mid B_j \in B, B_j \textit{ nonmaximal } B_i < B_j\}$.

The *constant set* $C(g, GC)$ for a goal g in a relaxable clause GC gives the goals in the body of GC that are preferred over g in establishing H, the head of GC. Thus, their truth has to be maintained while stating that the H-worlds where g is true are better than H-worlds where g is not true. However, we do not care if the goals in the constant set that were false become true in the process of moving to the better world. This motivates the division of $C(g, GC)$ into two complementary subsets C_{gp} and C_{gn}. The arbiter clause scheme for an atom g while proving H in the context of GC is then given by:

$$H \wedge (\bigwedge_{q_i \in C_{gp}} q_i) \wedge (\bigwedge_{q_j \in C_{gn}} \neg(q_j)) \wedge \neg g \rightarrow P_f(H' \wedge (\bigwedge_{q_i \in C_{gp}} q_i') \wedge g')$$

where the primed literals on the right hand side represent the fact that variables on the left hand side have been replaced by fresh variables. Thus, there are no variables in common between the left-hand sides and the right-hand sides. If the constant set $C(g, GC)$ for a goal g in a relaxable clause GC has n elements, then there are 2^n arbiter clauses corresponding to g because $C(g, GC)$ has 2^n subsets.

We generalize the above *program preference* translation for a single relaxable clause to the equivalent translation for a relaxable logic program, i.e., a finite set of relaxable clauses. Let GP be a relaxable logic program with n clauses $GC_1, \cdots, GC_n$. Let the translation of the relaxable clause GC_i give the clause set $S_i = \{C_{i1}, \cdots, C_{ik_i}\}$ of cardinality k_i, and the arbiter $\mathcal{A}_i$. The preference logic program π_{pref} corresponding to GP is given by

$$\mu \wedge (\bigwedge_j \Diamond(\pi_j \wedge \neg(\bigvee_{k \neq j} \pi_k))) \wedge \mathcal{A}$$

where μ is empty, $\mathcal{A}$ is the set-theoretic union of the various $\mathcal{A}_i$ and each π_i is obtained by picking one clause from each of the n clause sets given by the

S_i. Thus, the number of competing π_j is given by $\prod_{i=1}^{n} k_i$. The equivalent program preference logic program for our original example has $1 \leq j \leq 4$. The corresponding π_j are given by

$\pi_1 = \{p(X) \leftarrow q, r(X), s(X).\quad q.\quad s(2).\quad r(1).\}$
$\pi_2 = \{p(X) \leftarrow q, r(X).\quad q.\quad s(2).\quad r(1).\}$
$\pi_3 = \{p(X) \leftarrow q, s(X).\quad q.\quad s(2).\quad r(1).\}$
$\pi_4 = \{p(X) \leftarrow q.\quad q.\quad s(2).\quad r(1).\}$

The arbiter $\mathcal{A}$ is the arbiter $\mathcal{A}$ given above, since the arbiters for the other clauses are empty. We add distinct dummy predicates to the π_j so that the mutual-exclusiveness of the π_j is satisfied. The following is true.

Theorem 19. *Let π_p be the Horn preferential theory*

$$(\bigwedge_j \Diamond(\pi_j \wedge \neg(\bigvee_{k \neq j} \pi_k))) \wedge \mathcal{A}$$

If $1 \leq j \leq n$, then the preference modal structure that is the least model of π_p is $\langle \mathcal{W}, \mathcal{R}, \preceq, \mathcal{V} \rangle$ such that

1. *Cardinality of $\mathcal{W}$ is n,*
2. *$\mathcal{R}$ is the universal relation,*
3. *There exists $w_j \in \mathcal{W}$ such that $\mathcal{V}(w_j) = M_{\pi_j}$, where M_{π_j} is the least Herbrand model of π_j (over the language of π_p); and*
4. *$\preceq$ is the least relation on $\mathcal{W}$ induced by the arbiter $\mathcal{A}$;* i.e., *for any two worlds w_1 and w_2 $w_1 \preceq w_2$ iff there exists a ground instance of an arbiter clause*

$$\bigwedge_i B_i \rightarrow P_f(\bigwedge_j H_j)$$

such that $w_1 \models \bigwedge_i B_i$ and $w_2 \models \bigwedge_j H_j$.

At this point, a comparison between the *clause preference* and *program preference* translations is in order. We saw earlier that in the clause preference translation, the partial order on the bodies of relaxable clauses was captured *intensionally* by the *textual order* of the translated clauses. The program preference translation enables us to syntactically characterize these preference orders via the *arbiter*. Further, the richer modal preference models reflect these preference orders.

Lemma 20. *Let GP be a relaxable logic program, CP its clause preference translation and PP its program preference translation. Let M_{CP} be the least Herbrand model of CP and $M_{PP} = \langle \mathcal{W}, \mathcal{R}, \preceq, \mathcal{V} \rangle$ be the least modal preference model of PP. Then*

$$M_{CP} = \bigcup_{w_i \in \mathcal{W}} \mathcal{V}(w_i)$$

The partial order among the worlds, however, gives information about which solutions are preferred. The preference order motivates the following definition of an interpreter that computes the most preferred solutions.

Definition 21. Let GP be a goal preference logic program and $g(x)$ be a goal. Let $\mathcal{W}$ be the set of worlds in the least preference model of the program preference translation of GP. I is a **preferential interpreter** if $I(GP, g(x))$ is the set of all ground substitutions θ, such that there exists a world $w \in \mathcal{W}$ where $w \models (g(x))\theta$, and for every $w' \in \mathcal{W}$ with $w \preceq w'$ and $w' \not\preceq w$, either $w' \models (g(x))\theta$ or the extension of g at w' is empty. Note that $\mathcal{W}$ is the set of worlds in the least preference model of the program preference translation of GP.

Thus, the preferential interpreter would return $X = 1$ as the preferred solution for the goal $p(X)$ in our example program.

6 Conclusions

In this paper we motivated and presented a modal logic of preference and applied it to the problem of constraint relaxation in logic programming. The model-theoretic analysis of relaxable Horn clauses given in this paper has been extended to include a fix-point semantics as well as an operational semantics based on a variant of standard SLD-resolution. The details can be found in [14]. The applications of preference logic to knowledge representation and symbolic computing including formal planning, abduction and diagnosis are tremendous. Preference logics represent a first step towards bringing methods in artificial intelligence closer to classical methods in decision and utility theory. Deep connections between *preference* and *probability* exist [11] and they have to investigated in the context of our framework. Comparisons with *meta-logical* approaches to control of deductions also awaits analysis. We are confident that preference logics provide a unifying framework for the various philosophical logics used in artificial intelligence.

References

1. ACKERMANN, R. Comments on n. reshcer's semantic foundation for the logic of preference. In *The Logic of Decision and Action*. 1967.
2. BORNING, A., AND ET. AL., M. M. Constraint hierarchies and logic programming. In *Sixth International Conference on Logic Programming* (June 1989), pp. 149–164.
3. BROWN JR., A. L., MANTHA, S., AND WAKAYAMA, T. Preferences as normative knowledge: Towards declarative obligations. In *First International Workshop on Deontic Logic in Computer Science* (Amsterdam, The Netherlands, 1991), J. J. C. Meyer and R. J. Wieringa, Eds., pp. 142–164.
4. BROWN JR., A. L., MANTHA, S., AND WAKAYAMA, T. Preference logics and nonmonotonicity in logic programming. In *Logic at Tver, International Conference on Logical Foundations of Computer Science* (Tver, Russia, 1992), A. Nerode, Ed., Springer-Verlag.
5. BROWN JR., A. L., MANTHA, S., AND WAKAYAMA, T. Preference logics: Towards a unified approach to nonmonotonicity in deductive reasoning. In *Second International Symposium on Artificial Intelligence and Mathematics* (Ft. Lauderdale, Florida, 1992).

6. FISHBURN, P. Intransitive indifference in preference theory: A survey. *Operations Research 18* (1970).
7. HALLDEN, S. The logic of better. Lund, 1957.
8. HANSSON, B. Fundamental axioms for preference relations. *Synthese 18* (1968).
9. HANSSON, S. O. A new semantical approach to the logic of preference. *Erkenntnis 31* (1989), 1–42.
10. HUGHES, R. I. G. Rationality and intransitive preferences. *Analysis 40.3* (1980).
11. JEFFREY, R. C. *The Logic of Decision.* University of Chicago Press, Chicago, 1983.
12. LLOYD, J. *Foundations of Logic Programming.* Springer-Verlag, New York, 1984.
13. LUCE, R. D., AND RAIFA, H. *Games and Decisions.* 1957.
14. MANTHA, S. First-order preference theories and their applications. Tech. rep., Dept. of Computer Science, University of Utah, 1992.
15. VON WRIGHT, G. H. *The Logic of Preference.* University of Edinburgh Press, Edinburgh Scotland, 1963.
16. VON WRIGHT, G. H. The logic of preference reconsidered. *Theory and Decision* (1972), 55–67.

A Performance Evaluation of Backtrack-Bounded Search Methods for N-ary Constraint Networks

Pierre Berlandier

SECOIA Project, INRIA-CERMICS,
2004, Route des Lucioles, B.P. 93, 06902 Sophia-Antipolis Cedex

Abstract. An early study on the structural aspect of binary constraint problems has led to the definition of a backtrack bounded solving (BBS) method. In this paper, we apply this method to n-ary constraint problems and we try to weigh the benefits that can be expected from ' s use. Simple functions are described to implement BBS for acyclic n-ary problems and results of an experimental performance evaluation are given.

1 Introduction

Graphs are the most natural structures to represent and interpret constraint problems. Several research works focused on this structural aspect and led to the elaboration of efficient methods for solving sparse problems. The seminal ones are backtrack free [1] and backtrack bounded [2] search methods. They are regarded as attractive techniques by review papers such as [3] or [4]. However, they have seldom been implemented for experimental evaluation and they are most often ignored in the design of operational constraint interpretation systems. This lack of interest is surprising when they propose to solve some problems with a polynomial time complexity. Also noteworthy is the fact that all the structure based search methods (SBSM) were designed and remained in the restricted framework of binary constraint problems.

These observations prompted us to seek and weigh the true benefits that can be expected from the use of SBSM for the satisfaction of n-ary constraint problems. This paper presents our experiments and conclusions on this subject. We first review the existing relationships betwe.n the structure of the problem, its level of partial consistency and the complexity of finding the solutions, this in the framework of binary constraint problems. Then, we propose to map the ideas behind these relationships to n-ary constraint networks, which, in fact, represent the majority of real life problems. We describe simple functions to implement a backtrack bounded search for acyclic problems. Finally, we use these results to set an experimental comparison between the most common solving scheme (based on forward checking guided by the dynamic search rearrangement heuristics) and a solving scheme involving a SBSM.

2 Basics

The usual constraint formalism derives from [5]. It presents a constraint problem as a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R} \rangle$. $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the set of the n variables of the problem. $\mathcal{D}$ is a set of finite domains and D is a bijection from $\mathcal{X}$ to $\mathcal{D}$ such that $D(x_i)$ is the domain of x_i. $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of m constraints where c_i is a tuple (a pair, for binary problems) of variables $\langle x_{i_1}, \ldots, x_{i_p} \rangle$ from $\mathcal{X}$. Finally, $\mathcal{R}$ is a set of relations and R is a bijection from $\mathcal{C}$ to $\mathcal{R}$ such that $R(c_i)$ is the relation attached to c_i which defines the set of p-tuples allowed by this constraint.

The constraint satisfaction problem (CSP) consists in finding one or more substitutions V of all the variables in $\mathcal{X}$ such that:

$$\forall x_i \in \mathcal{X}, V(x_i) \in D(x_i) \text{ and } \forall c_i \in \mathcal{C} \ \langle V(x_{i_1}), \ldots, V(x_{i_p}) \rangle \in R(c_i)$$

Finding these globally consistent substitutions is NP-complete. The most common way to reduce the complexity is to apply a preliminary filtering step to the problem in order to enforce its partial consistency.

2.1 Partial Consistency

Enforcing partial consistency consists in finding and excluding some partial substitutions of $\mathcal{X}$ that are known to be incompatible with the constraint set. Inconsistencies can be detected at different levels by considering greater and greater subsets of the problem's variables.

A general consistency concept for binary constraint problems is k-consistency [1]. A problem is k-consistent if, given $k-1$ variables with consistent values, we can always complete this set with any other k-th variable of the problem along with a value in its domain so that all the constraints between these k variables are satisfied. A problem is said to be strongly k-consistent if it is consistent for every $j \leq k$.

2-consistency, also called arc-consistency (AC) [6] is mostly used in practice because it has empirically proven to offer the best tradeoff between the amount of search effort and problem simplification. Moreover, inconsistent substitutions being singletons, the arc-consistency process does only shrink the domain of the variables; it does not generate any new constraints.

In [2], the concept of k-consistency is extended to (i,j)-consistency which means that given i variables with consistent values, we can find values for j other variables such that the constraints between the $i+j$ variables are satisfied. Again, the problem is strongly (i,j)-consistent if it is (k,j)-consistent for every $k \leq i$. As for 2-consistency, $(1,j)$-consistency can be achieved simply by removing some values from the variables domain.

After partial consistency is enforced, finding global solutions to the CSP still entails a traversal of the search space, which is usually conducted by a backtrack search procedure. The instanciation order of the variables is a crucial parameter of the search. The following sections are concerned with finding an efficient one.

2.2 Backtrack Free Search

The works reported in [1] and then in [7] show that the complexity of the CSP is strongly related with the structure of its associated graph. As expected, the sparser is the graph, the easier is the problem. The results of these works have an important practical interest as they provide a mean to solve some problems in a reasonable polynomial time. They allow to determine the consistency level to install together with the sequence of variables that lead to a greedy instanciation process.

A central concept of constraint graph analysis is the width of a variable. In a given sequence of variables, it is defined as the number of variables that share a constraint with the variable in question and that precede it in the sequence. The width of a sequence is the maximum width of its variables and the width of a constraint graph is the minimum width of its possible sequences. Figure 1 shows three possible sequences for the binary constraint graph $\{\langle x_1, x_2\rangle, \langle x_1, x_3\rangle\}$ and their respective width (the instanciation proceeds top-down).

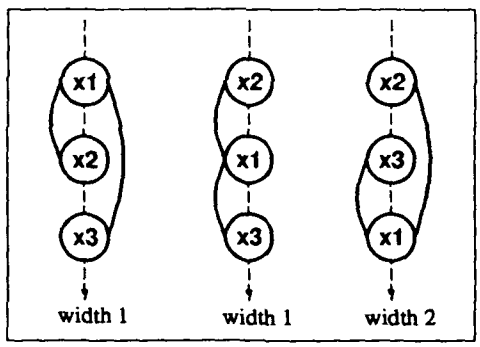


Fig. 1. Widths of different sequences for the same constraint graph.

Theorem 1 of [1] states that, when the level of strong consistency of a problem is greater than the width of its graph, there exists a sequence of variables that leads to backtrack free search. This result appeals to the intuition: when a problem is strongly k-consistent, it means that we can instanciate up to k variables of the problem without possible failure. Moreover, when a variable with width w is instanciated, we have to check the constraints it shares with at most w preceding variables, which is similar to the consistency check of $w+1$ variable values. Therefore, if we have $k > w$, we are sure that there is always a consistent choice for any variable in the sequence.

2.3 Directed Consistency

A tree is a graph of width 1. Indeed, if we assign an arbitrary orientation to the vertices and apply a topological sort to the resulting directed acyclic graph, we

obtain a sequence where each variable is preceded by at most one of its adjacent variable, that is a sequence of width 1. Solving a tree structured problem without backtracking thus entails strong 2-consistency.

Nevertheless, when having a closer look at it, we realize that full arc-consistency achieves more work than required. In fact, it is sufficient to enforce directed arc-consistency (DAC) following the order of instanciation [7]. This means that for two variables x_i and x_j such that x_i precedes x_j, it is sufficient that for all value $v_i \in D(x_i)$ there exists a compatible value $v_j \in D(x_j)$. The converse is not necessary as the value of x_j is always chosen after that of v_i and that there are no implicit constraints between v_i and v_j. Consider, for example, the leftmost sequence of Fig. 1 and suppose that arcs are representing equality constraints and that we have $D(x_1) = \{1,2,3\}$, $D(x_2) = \{0,1,2\}$ and $D(x_3) = \{2,3,4\}$. Getting directed arc-consistency costs 12 consistency checks and only reduces the domain of x_1 to $\{2\}$ whereas full arc-consistency costs 18 consistency checks and reduces all domains to that singleton. In the general case, when the worst-case complexity of AC is $O(md^3)$, the one of DAC is $O(nd^2)$ (where d is the size of the largest domain).

2.4 Backtrack Bounded Search

In [2], the width definition is extended to j-width which is the minimum of the widths of groups from 1 to j consecutive variables. As for the regular width, the j-width of a sequence is the maximum of the j-width of its variables and then, the j-width of the graph is the minimum j-width of its possible sequences. The characterization of backtrack free search is generalized to backtrack bounded search: when a graph is strongly (i,j)-consistent and when i is equal to the j-width of the graph, there exists an instanciation sequence so that backtracking is limited to $j-1$ instanciated variables.

Now, let b be the size of the maximal biconnected component of a constraint graph. It is showed in [2] that the $(b-1)$-width of this graph is 1. Therefore, the corresponding problem can be solved in time exponential in b, after its $(1,b-1)$-consistency has been enforced. This probably constitutes the most useful result of the backtrack bounded search study and also the inspiration source of our application to n-ary constraints, presented below.

3 N-ary Constraint Problems

So far, we have learned how to recognize easy binary constraint problems and how to solve them efficiently by first enforcing a certain level of (directed) consistency and then by following a given sequence of variables during instanciation.

However, real life problems are rarely expressed with binary constraints in their whole. Of course, it is always possible to translate a n-ary constraint problem into its binary equivalent [8], but this translation is often costly and results in a bigger problem which may be less tractable than the original one. That is why we wished to adapt the previous theorems on SBSM to n-ary constraint networks.

3.1 Principle

N-ary constraint problems are the straightforward extension of binary ones. Their representation is also a generalization of graphs, namely hypergraphs. Figure 2(a) shows the representation of a sample n ary constraint problem.

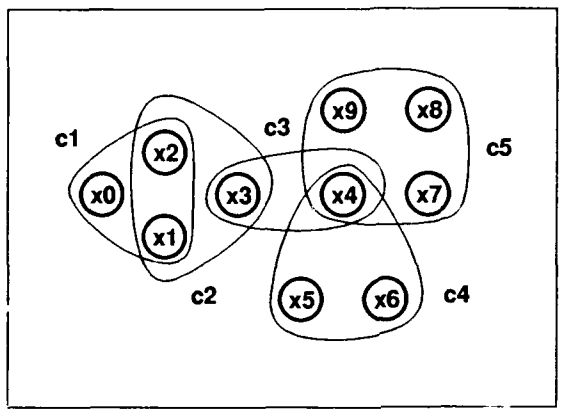


Fig. 2. A hypergraph of constraints.

For an acyclic constraint hypergraph $\mathcal{C}$, it is always possible to compute a sequence of constraints $\mathcal{S}$ such that any given constraint does share at most one variable with all the constraints that precede it in $\mathcal{S}$. Let us suppose that the following proposition holds:

$$\forall c_i \in \mathcal{C}, \forall x_{i_j} \in c_i, \forall v \in D(x_{i_j}), \exists \langle r_1, \ldots, r_p \rangle \in R(c_i), r_j = v \qquad (1)$$

Let c_1 be the first constraint of $\mathcal{S}$. The previous proposition implies that a consistent instanciation of its variables can be found. Now, let us suppose that all the variables of the constraints preceding a given c_i of $\mathcal{S}$ are consistently instanciated. From the definition of $\mathcal{S}$, we can assert that at most one variable of c_i is instanciated. Under this condition, proposition 1 implies that we can always find values for the non-instanciated variables of c_i so that it is satisfied.

This shows by recursion that a complete consistent instanciation can be found without backtracking across the variables of distinct constraints. The only backtracking that may occur is during the search for a consistent instanciation of the variables of a constraint. So, we can conclude that the amount of backtracking is restricted to $\|c_i\| - 1$ variables for each successive constraint c_i. The worst case complexity of the whole instanciation process is thus $O(md^a)$ where a is the maximal arity of the constraints in $\mathcal{C}$.

Moreover, the remark on directed consistency exposed in section 2.3 applies here again. The level of partial consistency described by proposition 1 is stronger than what is actually required. It is sufficient to make sure that:

$$\forall c_i \in \mathcal{S}, x_{i_j} = c_i \cap \{c_k \in \mathcal{S} \mid c_k \prec c_i\}, \forall v \in D(x_{i_j}), \exists \langle r_1, \ldots, r_p \rangle \in R(c_i), r_j = v$$

This consistency level can be achieved simply by filtering each constraint once, following the reverse order of sequence $\mathcal{S}$. The complexity of this process is also $O(md^a)$. Therefore, an acyclic n-ary constraint problem can be solved with complexity $O(md^a)$.

3.2 Implementation

In the general case, a constraint problem is not entirely arborescent. It is often composed of small treelike subproblems that are encircling one bigger cyclic subproblem. It is thus recommanded to combine the best search mechanism we know for the cyclic part with a SBSM for treelike parts. Moreover, provided that the cyclic part is likely to be the most difficult part of the problem, it should be solved first in order to be solved only once.

The constraint tree processing we propose can be implemented by three simple functions. The first one extracts the branches of the problem and returns a reverse topological sort of the constraints that compose those branches. It proceeds by stripping the leaves one by one from the graph, a leaf being a constraint that shares at most one attribute with the other constraints. This function can be defined as follows:

```
function strip(C)
let tree = ();
while C ≠ ∅
do let leaves = (c ∈ C | |c ∩ ⋃(C \ {c})| ≤ 1};
   if leaves = () then return tree
   else tree ← append(tree, leaves);
        C ← C \ leaves;
end strip
```

Once we have extracted the treelike parts, we can enforce their directed consistency. As we said before, this merely consists in filtering the constraints in the order returned by the strip function. Finally, and also from the sequence of constraints, we can build the sequence of variables to use for instanciation. This is the role of the following function:

```
function sequence(tree, C)
let sequence = ();
while tree ≠ ()
do let c = head(tree);
   tree ← tail(tree); C ← C \ {c};
   for all x ∈ c
   do if ¬∃c ∈ C, x ∈ c
      then sequence ← cons(x, sequence)
return sequence;
end sequence
```

In the hypergraph $\mathcal{C}$ of Fig. 2(a), the cyclic part is composed of c_1 and c_2 while the treelike part is composed of c_3, c_4 and c_5. Calling **strip**($\mathcal{C}$) returns either (c_4, c_5, c_3) or (c_5, c_4, c_3), which are functionally equivalent for our purpose. Then, calling **sequence**$((c_5, c_4, c_3), \mathcal{C})$ may return the sequence $(x_4, x_5, x_6, x_7, x_8, x_9)$ which conforms to the tree structure and can be used as a static instanciation order.

4 Experimental Evaluation

The goal of this experimental evaluation is to weigh the benefits that can be expected from the use of the tools presented in the previous section. As a reference for efficiency, we choose the best general solving scheme we know which combines backtracking with forward checking and follows a dynamic instanciation order (DO) termed dynamic search rearrangement. This ordering heuristics consists in selecting as the next variable to be instanciated the variable that has the smallest domain among the remaining ones. The literature [9, 10, 11] usually agrees that it gives the best overall performances for an average case analysis, experimentally as well as analytically.

Two quantities are of interest when benchmarking a constraint solving process: the number of constraint checks and the number of backtracks. As noted in [12], the first one is a good indicator of performance while the second one corresponds to the size of the exposed search space. What we present in our results is, for those two quantities, the difference between:

1. solving a problem using solely DO and solving it using a static tree ordering TO on treelike parts.
2. the same but preceded by a preprocessing step: AC alone in the first case and AC with DAC on treelike parts in the second one.

A positive difference is a point in favor of the SBSM whereas a negative one means it is inadequate.

4.1 Method

All the problems we are testing are built on the same model i.e. a central cyclic problem connected to several treelike parts. The chosen central problem is the well-known Zebra problem which has only one solution.

The treelike parts are varying in size and number. They are made of 1 to 5 constraints each and 1 to 8 trees are connected to the cyclic problem. Provided that the Zebra problem is composed of 62 constraints, the treelike parts are then representing from 8 to 65 percent of the total number of constraints in the problem. The variables of the treelike extensions are given the same domain as the ones of the Zebra problem.

For each experiment, the search for the first solution is repeated 50 times with a different initial permutation of the variables. The measure we use is the average of the results we obtain.

4.2 Results

Figures 3, 4 and 5 display two graphics each. The left one concerns constraint checks and the right one concerns backtracks. In each graphic, two curves are displayed. The black one represents the difference (AC, DO) − (DAC, TO) and the grey one represents the difference (DO) − (TO). The abscissa represents a combination of the size and number of treelike parts. It starts at size 1 for 1 to 8 trees, then size 2 also for 1 to 8 trees, etc. up to size 5.

- Figure 3 shows the results we obtained for trees made up of ternary constraints with a high fail probability (0.96). They are positive when solving without preprocessing but we should mention that the standard deviation (σ) of the measures for solving with DO is very high. For example, with 5 trees of size 3, the average count of consistency checks is 8858 but σ is 6546. This means that the performance depends on the choice of the first instanciated variable. If this variable belongs to a treelike part and due to the high filtering power of the constraints, the whole tree will be solved first and the solving of the cyclic part will be repeated several times, leading to disastrous results. On the contrary, if the first variable is chosen in the cyclic part, the latter will be solved first resulting in a good performance. So, here, tree ordering avoids dreadful mistakes.
  On the other hand, when preprocessing is applied, the results are not favorable to the SBSM any more. The preprocessing step reduces the domains so that the first variable can be chosen wisely.
- Figure 4 shows the results for an opposite example. Here, trees are made up of ternary constraints with a low fail probability (0.04). These results are chaotic, be it with or without preprocessing. Therefore, no definite conclusion can be drawn and by default, the simplest solution should be preferred (i.e. no use of the SBSM).
- Figure 5 shows the results for trees composed of constraints with a fail probability individually and randomly chosen between 0.04 and 0.96. Again, the results are such that no conclusion can be drawn.

Other experiments were made with various ranges of arity and satisfiability for constraints, domain size for variables or branching factor for trees. They all could be connected to the previous cases: when difficult constraints were dominating in the treelike parts, the results were near to those of Fig. 3.

5 Conclusion

We showed that a basic SBSM for n-ary constraint problems can be implemented easily and at low cost. Nevertheless, from the experimental results, we conclude that benefits can only be expected when the treelike parts of the problems are made of constraints with a high filtering power and that no preprocessing is applied. This indeed leaves a thin gap for the applicability of such a method.

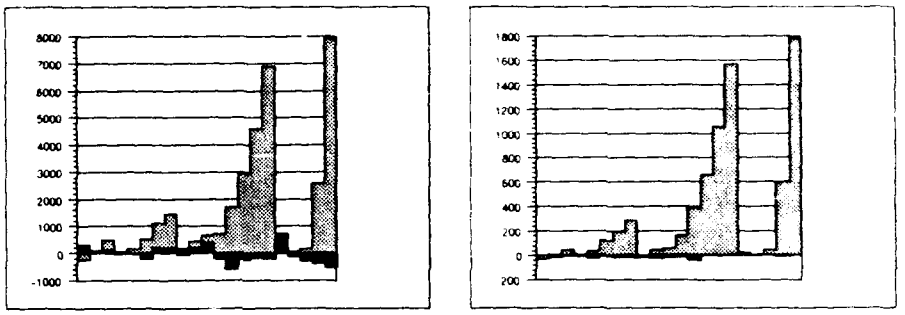


Fig. 3. constraints with high fail probability

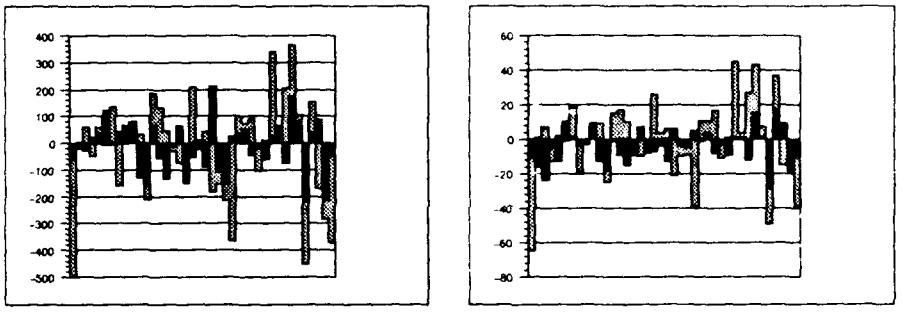


Fig. 4. constraints with low fail probability

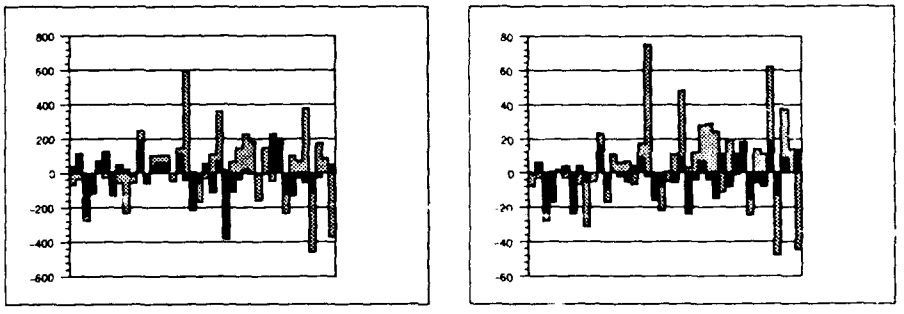


Fig. 5. constraints with random fail probability

References

1. E. Freuder. A sufficient condition for backtrack free search. *Journal of the ACM*, 29(1):24 32, 1982.
2. E. Freuder. A sufficient condition for backtrack bounded search. *Journal of the ACM*, 32(4):755 761, 1985.
3. P. Meseguer. Constraint satisfaction problems: An overview. *AICOM*, 2(1):3 17, 1989.
4. V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32 44, 1992.
5. U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7(3):95 132, 1974.
6. A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99 118, 1977.
7. R. Dechter and J. Pearl. Network based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1 38, 1988.
8. F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. Technical Report ACT-AI-222-89, MCC, 1989.
9. P. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117 133, 1983.
10. B. Nudel. Consistent labeling problems and their algorithms. *Artificial Intelligence*, 21:135 178, 1983.
11. J. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms*. PhD thesis, Carnegie Mellon University, 1979.
12. R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proc. IJCAI*, Detroit, Michigan, 1989.

Finite Domain Consistency Techniques: Their Combination and Application in Computer-Aided Process Planning

Manfred A. Meyer[1] and Jörg P. Müller[2]

[1] German Research Center for Artificial Intelligence (DFKI), Erwin-Schrödinger-Straße, 6750 Kaiserslautern, Germany
[2] German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany

Abstract. In this paper we present the weak looking-ahead strategy (WLA), a consistency technique on finite domains combining the computational efficiency of forward-checking with the pruning power of looking-ahead. We show that by integrating weak looking-ahead into PROLOG's SLD resolution we obtain a sound and complete inference rule, whereas standard looking-ahead itself has been shown to be incomplete. We outline how we use the weak looking-ahead technique for lathe tool selection in a CIM environment.

1 Introduction

Constraint Logic Programming has been shown to be a very useful tool for knowledge representation and problem-solving in different areas. Finite domain extensions of PROLOG together with efficient consistency techniques, such as forward-checking and looking-ahead, allow us to solve many discrete combinatorial problems efficiently by restricting the search space in an *a priori* manner. When using these consistency techniques to implement real-world applications, it turned out that forward-checking and looking-ahead, as provided in most finite-domain PROLOG extensions, are not totally satisfactory: Forward-checking often does not give any pruning at all until variables become singletons, whereas standard looking-ahead forces strong pruning of the search-space but induces serious control overhead.

In this paper we present a consistency technique on finite domains, which combines the efficiency of forward-checking and the pruning power of standard looking-ahead: The basic idea of this **weak looking-ahead (WLA)** strategy is to apply looking-ahead only once to a constraint and to use forward-checking for further restricting the domains of its arguments. The way weak looking-ahead came into being stood in a close relation to a real-life application: In the ARC-TEC project at DFKI we have been developing a knowledge-based system generating workplans for lathe CNC machines. (μCAD2NC, [2]). After we decided to solve the subproblem of selecting appropriate lathe tools for the various processing steps by using constraints, we experimented with several consistency algorithms. Soon we realized that on the one hand, forward-checking is too weak

for some applications where an earlier pruning of the search space is desired. On the other hand, using looking-ahead for this application is a bit like breaking the butterfly on the wheel. These observations entailed the wish for a new technique which causes only a little more cost than forward-checking, but which can achieve much better pruning results in many cases (see the example in section 4).

2 Finite Domain Consistency Techniques

Over the past years, increasing attention has been paid to using constraints in logic programming [4, 11] for it presents a very powerful incorporation of the advantages both of logic programming (declarativity, relational form, nondeterminism) and consistency techniques for constraint solving problems [7]. The use of consistency techniques allows one to overcome the basic shortcomings of logic programming languages, which are mainly caused by their poor, mostly backtracking-like control strategies. Techniques such as forward-checking and looking-ahead are used to restrict the domains of variables in an active manner and to achieve an *a priori* pruning of the search space.

In this section we will give a short informal description of both forward-checking and looking-ahead according to [11]. In section 3 we present the weak looking-ahead method which is essentially based on these techniques.

Forward-Checking: The idea of forward-checking is formally expressed by the forward-checking inference rule (FCIR). Informally, a constraint C can be used in a forward-checking manner as soon as all except one of its domain-variable arguments, say X, are instantiated to a ground value. Then, C is called *forward-checkable*. C can be considered a unary predicate $C'(X)$, and the set of possible values that can be given to X can be restricted to those elements a satisfying $C'(a)$.

Forward-checking has turned out to be one of the most popular consistency techniques, since it can be easily implemented (cf. [10]), it yields reasonable pruning results for many applications, keeping the computational costs fairly low, and since there exist sound and complete proof procedures based on normal SLD resolution combined with forward-checking (see [11]).

The main drawback of forward-checking is its strong applicability precondition: A predicate can be executed by the FCIR only if all except one of its variables are instantiated to a ground value. Thus, for predicates with many arguments and/or many variables, at a given point of computation, there is only a relatively small probability that forward-checking can be applied to it. Moreover, especially when computation starts, it is very often the case that no constraint is forward-checkable. That means that choices have to be made, i.e. variables are instantiated in a more or less random manner. Thus, the devil of backtracking which we would like to exorcize by the use of consistency techniques, returns through the back door. Finally, some constraints, such as $=$, $>$, and $<$ should

not be executed by forward-checking at all, because they embody a great deal of structural information about the relation between their arguments[3].

Looking-Ahead: The looking-ahead strategy [6] offers a powerful possibility to reduce the number of values that can be assigned to variables of a constraint, even if this constraint is not yet forward-checkable.

For every domain variable X appearing as an argument of an N-ary constraint C, and for every value within the domain of X, it must be checked whether there exists at least one admissible value from the domain of each domain variable Y appearing in C so that the constraint C is satisfied. The arguments of C which are not domain variables must be ground.

By using the Looking-Ahead Inference Rule (LAIR), the search space can be pruned at an early stage of computation. However, the trouble with standard looking-ahead is that it is a very expensive method of ensuring arc-consistency. Therefore, for most applications it is considered inappropriate [3]. Nevertheless, it would be a shame to forgo all the benefits brought about by the strong pruning capabilities of looking-ahead.

In the next section, we present *weak looking-ahead*, which can be regarded as a compromise between forward-checking and looking-ahead. Assume e.g. that, in our looking-ahead example above, we would perform the first looking-ahead step as shown, but after that, we would *not* do any more looking-ahead, but instead solve the (now simplified) problem by normal resolution or by forward-checking. This procedure expresses the main idea of the weak looking-ahead strategy which we will discuss in more detail in the following.

3 The Theoretical Background of WLA

The basic theoretical work in the area of using consistency techniques in logic programming has been done by van Hentenryck [11]. This research has contributed a great deal to both forming a solid framework and preserving the logic part of the programming languages developed while achieving a much better control behaviour than that achieved by standard logic programming languages such as PROLOG (cf. [5]).

In this section, we will give a formal definition of the weak looking-ahead inference rule, and we will present the basic formal properties such as soundness and completeness of the proof procedure defined on top of WLA. The terminology we use and the sense we use it are basically the same as in [11].

The weak looking-ahead strategy combines the use of LAIR and FCIR. A similar technique has been informally proposed in [3] as "first-order looking-ahead". We present a generalized technique we call weak looking-ahead. This name seems more appropriate for expressing what the underlying algorithm really does. The basic idea of WLA is that each constraint can be selected by

[3] For example, the information that two variables X and Y are equal should not only be used if X or Y are ground. Rather, the equality constraint should be maintained from the moment it has been stated (see [10]).

the looking-ahead part *not more than once*, and that this should happen at an appropriate time. After this, only the FCIR (or normal inference) can be applied to it. This idea is covered by the following definitions.

Definition 1 (WLA-checkable). An atom $p(t_1, ..., t_n)$ is called *WLA-checkable* if p is a constraint and

- $p(t_1, \ldots, t_n)$ is lookahead-checkable and has not yet been selected by WLA, or
- $p(t_1, \ldots, t_n)$ is forward-checkable and has already been selected by WLA.

Definition 2 (WLA). Be P a program, $G_i = \ ?- \ A_1, \ldots, A_k, \ldots, A_m$ a goal and σ_{i+1} a substitution. G_{i+1} is derived by WLA along with σ_{i+1} from G_i and P if the following holds:

1. A_k is WLA-checkable, with $x_1, \ldots, x_n$ being WLA variables in A_k.
2. If A_k is lookahead-checkable and WLA has not been applied to A_k in the actual proof, then goto 3; otherwise goto 7.
3. For each x_j, the new domain e_j is
   $e_j = \{v_j \in d_j \mid \exists v_1 \in d_1, \ldots, v_{j-1} \in d_{j-1}, v_{j+1} \in d_{j+1}, \ldots, v_n \in d_n$ such that $\sigma(A_k)$ is a logical consequence of P, $\sigma = \{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n\}\} \neq \emptyset$.
4. y_j is the constant c if $e_j = \{c\}$, or a new variable which ranges over e_j, otherwise.
5. $\sigma_{i+1} = \{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$.
6. G_{i+1} is either $?- \ \sigma_{i+1}(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_m)$ if at most one y_j is a domain variable, or $?- \ \sigma_{i+1}(A_1, \ldots, A_m)$, otherwise. EXIT.
7. A_k is forward-checkable, let x_d be the forward variable inside A_k. Then the new domain e is defined as $e = \{a \in d \mid P \models A_k\{x_d \leftarrow a\}\} \neq \emptyset$.
   σ_{i+1} is defined as $\sigma_{i+1} = \{x_d \leftarrow c\}$, if $e = \{c\}$, where y_e is a new domain variable, otherwise. $G_{i+1} = ?- \ \sigma_{i+1}(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_m)$.

The main point of the above definition is point 6, which uses SLDFC-resolution (SLD-resolution extended by the FCIR), whose soundness and completeness have been proven, in order to finish the proof after some prepruning has been done by using the LAIR in a definite way. Thus, if we want to prove soundness and completeness of WLA, we basically have to check the LAIR part. Since this part gets involved not more than once for each goal, and since this happens as early as possible (due to point 2 of the definition), the disadvantages of the LAIR such as its incompleteness and the high computational overhead can be avoided.

Proposition 3 (Soundness of WLA). *Let P be a program and G_i be the goal $?- \ A_1, ..., A_k, ..., A_m$ with A_k WLA-checkable. Let the goal G_{i+1} be derived by WLA along with σ_{i+1} from G_i, P as $G_{i+1} = \ ?- \ \sigma_{i+1}(A_1, ..., A_{k-1}, A_{k+1}, ..., A_m)$. G_i is a logical consequence of P iff G_{i+1} is a logical consequence of P.*

The next result concerns the completeness of WLA. This means that we can define a complete proof procedure using weak looking-ahead.

Definition 4 (SLDW-resolution). A first-order resolution proof procedure is called *SLDW-resolution*, if it uses weak looking-ahead for WLA-checkable goals and normal SLDD-derivation (extended SLD-resolution based on domain-variable unification) for other goals.

We can prove the completeness of such a proof procedure by making use of the completeness of the FCIR, showing that by applying the LAIR not more than once to each goal, no solutions are lost. The completeness result is expressed by the following proposition.

Proposition 5 (Completeness of WLA). *P be a logic program, G be a goal. If there exists an SLDD-refutation of $P \cup \{G\}$, then there also exists an SLDW-refutation of $P \cup \{G\}$. Moreover, if σ is the answer substitution from the SLDD-refutation of $P \cup \{G\}$, and ρ is the answer substitution from the SLDW-refutation of $P \cup \{G\}$, then $\rho \leq \sigma$.*

For the proofs of propositions 3 and 5 we refer to [10].

4 Using Weak Looking-Ahead for Tool Selection

In this section, we will show the usability of the weak looking-ahead inference rule by an example from the concrete domain of the ARC-TEC project [1] at DFKI which constitutes an AI approach to implement the idea of computer-integrated manufacturing (CIM). For its evaluation, an expert system for production planning has been developed.

4.1 The Lathe-Tool Selection Problem

The application problem we are dealing with for the rest of this paper will be to find appropriate lathe tools to manufacture a given workpiece. Depending on the shape, the material and other attributes of the lathe part to be manufactured, the work-plan consists of a number of different steps. A typical work-plan may provide one step for roughing, another step for finishing and a third (facultative) step for doing the fine finishing of the lathe part. For each processing step, appropriate tools have to be chosen.

This tool selection heavily depends on a lot of geometrical (e.g. the edge-angle) as well as technological parameters (e.g. material, process etc.). Moreover, the tool system itself consists of subparts which have to be combined, e.g. the tool holder, the material of the plate and its geometry. In practice, there are a lot of restrictions, e.g. which holder to use for which plate, or which kind of plate geometry to use for which workpiece contour.

To keep things simple, we assume that a lathe tool consists of two basic parts: the *cutting plate*, which actually cuts the material, and the *tool holder*, which serves to hold the cutting plates. In our application, we are now concerned with finding a well-suited tool—or rather: a number of well-suited tools—starting from a set of constraints which describe the current problem. Lathe-tool selection

then results in a set of possible holder/tool combinations for each skeletal plan or manufacturing feature.

When formalizing the tool selection problem as a CSP, the first thing we have to do is to restrict the number of input parameters. For our small example we will use the variables **Holder** and **Plate**, denoting the tool holder and the cutting plate, **Process** and **WP-material**, denoting the actual kind of processing and the material of the lathe workpiece, respectively. Furthermore, we will use three variables *denoting* different angles, namely **Beta-max**, **Edge-Angle**, and the tool-cutting edge-angle **TC-Edge-Angle**. These angles are denoted by β, ε, and χ, respectively, in figure 1.

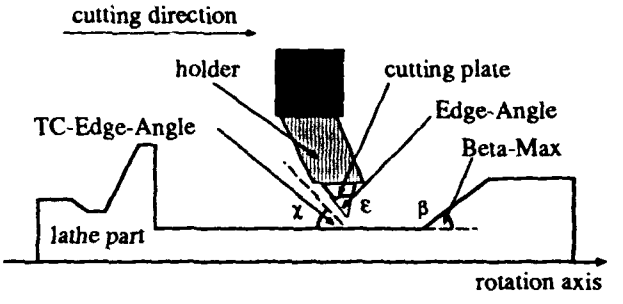


Fig. 1. The Angle *Constraint*

Having identified the problem variables, the constraints can be put on the variables. In the following, we will consider only the most important constraints, i.e. **holder_tcea(Holder, TC-Edge-Angle)** to describe the functional relation between a holder and its tool-cutting edge-angle; **plate_ea(Plate, Edge-Angle)** to denote that each plate has its own edge-angle; **compatible(Holder, Plate)** to express the compatibility condition between tool holders and cutting plates; **hard_enough(Plate, WP-Material)** to describe that for materials with different degrees of hardness, different *cutting*-plates have to be used; **process_holder(Process, Holder)** to denote appropriate types of holders for the different steps of processing; **process_edge_angle(Process, Edge-Angle)** to express the rule of thumb that for roughing, plates with big edge-angles should be chosen, whereas for finishing, smaller edge-angles are appropriate; finally, **TC-Edge-Angle + Edge-Angle + Beta-Max** shall be **less than 180°**. This constraint becomes evident when looking at figure 1, where the angles are denoted by χ, ε, β, respectively.

4.2 A FIDO Program for Tool Selection

In this subsection, we propose a formalization and solution of the tool-selection problem using FIDO, a **FI**nite **DO**main PROLOG extension developed at DFKI (see [9] for a survey). We will process the constraints defined above by using *weak looking-ahead* (WLA) and *forward-checking* consistency techniques. We will give

a trace of the constraint propagation process and show the advantages of WLA compared to standard looking-ahead or forward-checking techniques.

```
tool_sel(Holder, Plate, Process, Material, Beta-Max, Edge-Angle, TC-Edge-Angle) :-
    \* definition of the domains and domain variables *\
    define_domain(holders, [Holder], [tmaxp-ptl1, tmaxp-ptl2, tmaxp-ptl3]),
    define_domain(plates, [Plate], [dnmm-71, vnmm-71, cnmm-71, tnmm-71, dnmm-41,
      vnmm-41, cnmm-41, tnmm-41]),
    define_domain(processes, [Process], [roughing, finishing, fine-finishing]),
    define_domain(materials, [Material], [steel, cast, alu]),
    define_domain(angles, [Beta-Max, Edge-Angle, TC-Edge-Angle], 0..90),
    \* dynamic building of the constraint net *\
    wla(holder_tcea(Holder, TC-Edge-Angle)), \* (1) *\
    wla(plate_ea(Plate, Edge-Angle)), \* (2) *\
    wla(TC-Edge-Angle + Edge-Angle + Beta-Max < 180), \* (3) *\
    forward(compatible(Holder, Plate)), \* (4) *\
    forward(hard_enough(Plate, Material)), \* (5) *\
    forward(process_holder(Process, Holder)), \* (6) *\
    forward(process_edge_angle(Process, Edge-Angle), \* (7) *\
    \* instantiate variables using a first-fail instantiation predicate *\
    instantiate_dl([Holder, Plate, Process, Material, Beta-Max, Edge-Angle, TC-Edge-Angle])
```

Fig. 2. *A FIDO Program for Tool Selection*

```
\* definitions of the constraints: *\

\* (1) *\
  holder-tcea(tmaxp-ptl1, 80).
  holder-tcea(tmaxp-ptl2, 60).
  holder-tcea(tmaxp-ptl3, 45).
  ...
\* (2) *\
  plate-ea(cnmm-71, 80).
  plate-ea(tnmm-71, 60)
  plate-ea(dnmm-71, 55).
  plate-ea(vnmm-71, 35).
  ...
\* (4) *\
  compatible(tmaxp-ptl1, cnmm-71).
  compatible(tmaxp-ptl1, tnmm-71).
  compatible(tmaxp-ptl1, dnmm-71).
  compatible(tmaxp-ptl1, vnmm-71).
  ...

\* (5) *\
  hard_enough(cnmm-71, cast).
  hard_enough(tnmm-71, cast).
  hard_enough(dnmm-71, cast).
  hard_enough(vnmm-71, cast).
  hard_enough(cnmm-41, cast).
  hard_enough(tnmm-71, cast).
  hard_enough(dnmm-71, cast).
  ...
\* (6) *\
  process_holder(roughing, tmaxp-ptl1).
  process_holder(roughing, tmaxp-ptl2).
  process_holder(roughing, tmaxp-ptl3).
  ...
\* (7) *\
  process_edge_angle(roughing, Edge-Angle) :-
      forward(Edge-Angle < 65),
      !.
  ...
```

Fig. 3. The Tool Selection Database

In the following we will show how the program behaves in a concrete example. For the purpose of referencing the constraints, we numbered them consecutively from 1 to 7 in figure 3. Assume that the following call is performed:

```
?- tool_sel(Holder, Plate, roughing, cast, 70, TC-EA, EA).
```

This call to the tool-selection program provides the following input data: It binds the variables **Process**, **Material**, and **Beta-Max** to **roughing**, **cast**, and **70°**, respectively. The variables **Holder**, **Plate**, **TC-EA** and **EA** are supposed to be the output variables for this call. Now let us see how the call is processed:

1. First, the constraint **holder_tcea** (constraint number 1) is executed in a weak looking-ahead manner. Whereas the holder domain is not changed at all, the domain of the variable **TC-Edge-Angle** is restricted to the value {**45, 60, 80**}. Now, the constraint is reformulated as a forward-checking constraint and is suspended, since it is actually not forward-checkable.
2. The same as described for constraint #1 happens to constraint #2: whereas the variable **Plate** is left unchanged, the domain of the variable **Edge-Angle** is restricted to {**35, 55, 60, 80**}, and the forward-checking version of the constraint is suspended until later.
3. Now, constraint #3 is about to be executed. This is the constraint which actually benefits most from the WLA control. By performing the lookahead part of the weak looking-ahead algorithm, the domains of the variables **TC-Edge-Angle** and **Edge-Angle** are restricted to {**45, 60**} and {**35**}, respectively. The **Edge-Angle** variable becomes instantiated. Since constraint #3 is suspended as **forward(TC-Edge-Angle < 75)**, the instantiation of **Edge-Angle** causes constraints #2 and #7 to fire.
4. Constraint #2 is woken up and restricts the **Plate** variable to the values {**vnmm-71, vnmm-41**}. Then, constraint #7, both of whose variables are instantiated, is checked successfully. Thus, constraints #2 and #7 are done.
5. Constraint #5 is forward-checkable, since the workpiece material has been determined by the call to the main goal **tool_sel**. Since **vnmm-41** is not suitable for processing cast iron, the **Plate** domain becomes restricted to the set {**vnmm-71**}. Thus, **Plate** is instantiated to its singleton value **vnmm-71**.
6. Due to the instantiation of **Plate**, constraint #4 becomes forward-checkable, it is woken up and restricts the **Holder** domain to the values {**tmaxp-pt11, tmaxp-pt12**}.
7. Now, constraint #6 is checked. Because of the initialization of the variable **Process** to {**roughing**}, the constraint is forward-checkable. However, since both holders are appropriate for roughing, there is no further restriction of the **Holder** domain.

At this intermediate stage, the values of the variables are as follows:
Holder = {**tmaxp-pt11, tmaxp-pt12**}, **Plate** = **vnmm-71**, **Material** = **cast**, **Process** = **roughing**, **Edge-Angle** = **35**, and **TC-Edge-Angle** = {**45, 60**}.

Then, the instantiation predicate **instantiate_dl** (instantiation with first-fail heuristics using the domain lengths) is called.

8. The variable **Holder** is instantiated to the first element of its domain, which is **tmaxp-pt11**. This instantiation wakes up constraint #1, which fails because a tool-cutting edge-angle of **80°** is no longer allowed. Thus, for the first time, backtracking is necessary.
9. On backtracking, **Holder** is instantiated to the last remaining value, namely **tmax-pt12**. This wakes up constraint #1 and instantiates the **TC-Edge-Angle** variable to the value **60**.

Finally, we have found an admissible pair (**tmaxp-pt12, vnmm-71**) consisting of a holder and a cutting plate which satisfy the initial constraints. The solution

could be achieved by making only two choices, including one choice made by backtracking.

Let us now evaluate the program behaviour shown above. Here, we would like to stress the effects of using weak looking-ahead rather than forward-checking or standard looking-ahead for some of the constraints in our example. First, it is clear that weak looking-ahead yields much better pruning results than forward-checking. For instance, by using a weak-looking-ahead (*wla*) instead of a *forward* declaration for the constraints #1 to #3 in our example, we could immediately and largely restrict the domains of several variables, whereas if using a forward-checking control we would have had to wait until one of the constraint variables would have been instantiated.

Secondly, we will have to answer the question as to the advantages of weak looking-ahead over standard looking-ahead. By applying the LAIR not more than once for each constraint, the high expense of re-checking whether it can be applied at a later stage of computation can be avoided. Thus, some work has to be done, but it only has to be done once. This especially concerns constraint #3. After applying the LAIR to it, all we have to do in the following is to check whether a value will be assigned to any of its variables. Using looking-ahead, we would have had to pay attention to *each* time a value is removed from the domain of one of the constraint variables, and do a new looking-ahead check then. Thus, applying the LAIR only once and continuing with a forward-checking execution of the constraint allows the construction of a sound and complete proof procedure, as pointed out in section 3. It allows a combination of the simplicity of forward-checking with the power of looking-ahead, however, avoiding the high computational cost of the latter.

Finally, we would like to consider *when* the looking-ahead step in WLA should be done. In our experience, useful control strategies should perform the step as early as possible, or do it for a constraint as soon as it has been touched the first time, i.e. as soon as one of its constraint variables has been restricted.

5 Conclusions

Finite domain consistency techniques like forward-checking and looking-ahead contribute to make constraint logic programming an appropriate tool for expressing and solving a rich class of combinatorial problems. When applying these techniques in a real-world application, the need for a combination of forward-checking and looking-ahead came up. In this paper, we have presented the *weak looking-ahead* technique which combines the pruning power of looking-ahead with the efficiency of forward-checking. We have shown that the resulting inference rule is sound and complete. Finally, we have shown how to use weak looking-ahead for lathe tool-selection in a CIM environment. The language FIDO which we have used for this application has been developed at DFKI and will further be extended to support hierarchically structured domains and to make use of their structure for both expressing and applying constraints over them [8].

Acknowledgements

The work presented in this paper has been done within the ARC-TEC project at DFKI and was partly supported by the German Federal Ministry of Research and Technology (BMFT) under grant ITW 8902 C4.

We would like to thank Jane Bensch, Hans-Günther Hein and Eva Völker for proofreading earlier versions of this paper. The three anonymous referees also provided useful comments on the organisation of the paper.

References

1. A. Bernardi, H. Boley, K. Hinkelmann, P. Hanschke, C. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, G. Schmidt, F. Schmalhofer, and W. Sommer. ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. In *Expert Systems and their Applications: Tools, Techniques and Methods*, Avignon, France, 1991. Also available as Research Report RR-91-27, DFKI GmbH, P. O. Box 2080, D-6750 Kaiserslautern.
2. H. Boley, P. Hanschke, M. Harm, K. Hinkelmann, T. Labisch, M. Meyer, J. Müller, T. Oltzen, M. Sintek, W. Stein, and F. Steinle. μCAD2NC: A declarative lathe-workplanning model transforming CAD-like geometries into abstract NC programs. Technical Report D-91-15, DFKI GmbH, P. O. Box 2080, D-6750 Kaiserslautern, November 1991.
3. D. de Schreye, D. Pollet, J. Ronsyn, and M. Bruynooghe. Implementing Finite-Domain Constraint Logic Programming on Top of a PROLOG-System with Delay-mechanism. In N. Jones, editor, *Proc. ESOP'90*, pages 106–117, 1990.
4. J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. POPL-87*, Munich, Germany, 1987.
5. J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) Language and System. Technical Report CMU-CS-90-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, October 1990.
6. A.K. Mackworth. Consistency in Networks of Relations. *AI Journal*, 8(1):99–118, 1977.
7. P. Meseguer. Constraint Satisfaction Problems: An Overview. *AICOM*, 2(1):3–17, 1989.
8. M. Meyer. Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment. In *Fifth International Symposium on Artificial Intelligence*, pages 167–177. AAAI Press, December 1992.
9. M. Meyer, H.-G. Hein, and J. Müller. FIDO: Finite Domain Consistency Techniques in Logic Programming. In A. Voronkov, editor, *Logic Programming: Proceedings of the 1st and 2nd Russian Conferences*, volume 592 of *LNAI*, pages 294–301. Springer-Verlag, Berlin, Heidelberg, 1992.
10. J. Müller. Design and Implementation of a Finite Domain Constraint Logic Programming System based on PROLOG with Coroutining. Diploma thesis, Computer Science Department, University of Kaiserslautern, 1991. Also available as Technical Report D-91-02, DFKI GmbH, P. O. Box 2080, D-6750 Kaiserslautern.
11. P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, 1989.

Should Decision Trees be Learned from Examples or from Decision Rules?

Ibrahim F. Imam and Ryszard S. Michalski

Center for Artificial Intelligence
George Mason University
iimam@aic.gmu.edu & michalski@aic.gmu.edu

ABSTRACT

A standard method for determining decision trees is to learn them from examples. A disadvantage of this approach is that once a decision tree is learned, it is difficult to modify it to suit different decision making situations. An attractive approach that avoids this problem is to learn and store knowledge in a declarative form, e.g., as decision rules, and then, whenever needed, generate from it a decision tree that is most suitable in any given situation. This paper describes an efficient method for this purpose, called AQDT-1, which takes decision rules generated by the learning system AQ15 and builds from them a decision tree optimized according to a given quality criterion. The method is able to build conventional decision trees, as well as the so-called "skip noder" trees, in which measuring attributes assigned to some nodes may be avoided. It is shown that "skip-node" trees can be significantly simpler than conventional ones. In the experiments comparing AQDT-1 with C4.5, the former outperformed the latter both in terms of the predictive accuracy as well as the simplicity of the generated decision trees.

Key words: machine learning, inductive learning, decision trees, decision rules.

1. Introduction

Methods for learning decision trees from examples have been quite popular in machine learning due to their simplicity. Decision trees built this way can be quite efficient, as long as the decision making situations for which they are optimized remain relatively stable. Problems arise when these situations change, or the assumptions under which the tree was built do not hold. For example, in some situations it may be very difficult to determine the value of a certain attribute on the path from the root. One would like to avoid measuring this attribute, and still be able to classify the example. If the cost of measuring of various attributes changes, it is desirable to restructure the tree so that the "inexpensive" attributes are evaluated first. A restructuring is also desirable if there is a significant variation in the frequency of occurrence of examples from different classes. Restructuring a decision tree is, however, difficult because the tree represents a form of procedural knowledge.

An attractive alternative that avoids the above problems is to learn and store knowledge in the form of decision rules, and to generate from them an appropriate decision tree "dynamically," as needed. Decision rules represent knowledge declaratively, and thus do not impose any order on the evaluation of the attributes. Since the number of rules is typically much smaller than the number of examples, generating decision trees from

rules can potentially be very fast. This way, one can always generate a tree that is tailored to the specific decision situation. For example, one may be able to generate a decision tree that avoids evaluating an attribute that is difficult or impossible to measure, or a decision tree that fits well some particular distribution of the decision classes. In some situations, it may be unnecessary to generate a complete decision tree, but instead only the part with the leaves associated with the decision classes of interest. The proposed approach would generate only the desirable part. This could be interpreted as a generation of "virtual" decision trees.

A disadvantage of this approach is that it requires determining decision rules first. There are, however, very efficient methods for generating decision rules. Also, the rules need to be generated only once, and then can be used many times for generating any type of decision tree according to various decision making situation.

This paper presents a simple and efficient method for generating decision trees from decision rules. The method employs the AQ algorithm for generating rules. It also reports results from experiments comparing it with a well-known method for learning decision trees from examples, implemented in the C4.5 program.

2. Generating Decision Trees from Decision Rules: AQDT-1

Decision trees are normally generated from examples of decisions. The essential aspect of any method for this purpose is the *attribute selection criterion* that is used for choosing attributes to be assigned to the nodes of the tree. Among well-known criteria are the entropy reduction measure (e.g., Quinlan, 1979), the gini index of diversity (Breiman, et al., 1984), and others (Cestnik & Karalic, 1991; Mingers, 1989).

The first algorithm for generating decision trees from examples was proposed by Hunt, Marin and Stone (1966). This algorithm was subsequently modified by Quinlan (1979, 1983), and then improved and/or applied by many authors to a variety of learning problems (e.g., Quinlan, 1983; Breiman, et al., 1984). Later, Quinlan (1986), and Bratko and Kononenko (1987) added "tree pruning" procedures to handle data with noise.

In contrast to the above, the proposed method generates decision trees from decision rules. The method, called AQDT-1 (AQ-based Decision Trees), assumes that decision rules are generated from examples by the inductive learning system AQ15 (Michalski et al., 1986). AQDT-1 uses an attribute selection criterion that is based on the properties of *the rules*, rather than the properties of the *training examples*. One rule may describe a large number of examples. Decision rules are more powerful knowledge representation than decision trees, because they can directly represent any description in disjunctive normal form, while the latter can represent directly only a disjunctive normal form in which all conjunctions are mutually disjoint. Therefore, when transforming a set of arbitrary decision rules into a decision tree, one faces an additional problem of handling logically intersecting rules (conjunctions).

The proposed method for solving the first problem, i.e., choosing attributes on the basis of the properties of rules, employs a measure of "utility" of an attribute for reducing a given set of rules and some other criteria. The second problem (handling non disjoint rules) can be resolved by introducing additional nodes in the decision tree, or by assigning a "Don't care" value to some branches. This value serves as a connection edge between subtrees corresponding to non-disjoint rules. The branch with such a "value" is traversed when one does not know, or wants to ignore the value of the attribute assigned to the node from which it stems. Decision trees that have nodes with "don't care" values are called, for short, "skip-node" trees.

The input to the program AQDT-1 consists of rules generated by AQ15. Each such rule represents a conjunction of conditions. AQDT-1 creates a data structure for each concept description (a set of rules). This structure has fields such as the number of rules, the number of conditions in each rule, and the number of attributes in the rules. The system also creates an array of attribute descriptions. Each attribute description contains the attribute's name, domain, type, the number of legal values, a list of the values, the number of rules that contain that attribute, and values of that attribute for each rule. The attributes are arranged in the array in the a lexicographic order, first in the descending order of the number of rules that contain that attribute, and second, in the ascending order of the number of the attribute's legal values. AQDT-1 constructs a decision tree from decision rules by recursively selecting the "best" attribute at a given step, and assigning it to the new node. The difference between building a decision tree from rules and building it from examples is that in the former, attributes are selected on the basis of the role the attributes play in the rules, rather than on the basis of the coverage of examples, as in learning from examples.

The criterion for attribute selection should reflect the decision making situation. For example, if measuring different attributes involves different costs, then these costs should be included in the criterion. If some decision classes occur much more frequently than others, the criterion should favor measuring the attributes that occur in the rules for these classes. To be able to compare the AQDT-1 method with the C4.5 program, we assume here a selection criterion that is oriented toward producing trees with the minimum number of nodes. This criterion is composed of three elementary criteria: 1) the total *attribute utility*; 2) the number of different attribute values in the rules; and 3) the number of rules that contain the given attribute. The total attribute utility is the sum of the *class utilities*—the utilities of the attribute for each decision class. Suppose that decision classes are C1, C2,..., Cm. Suppose further that V1, V2,....., Vn are sets of values of some attribute A that occurr in rules for classes C1, C2, ...Cm, respectively. The utility of A for Class Ci, U(A, Ci), is the number of sets Vj, (j=1,...,m, and j≠i) that are disjoint with Vi. plus 1. The total utility, U(A), of the attribute A is:

$$U(A) = \sum_{i=1}^{m} U(A, Ci) \qquad (1)$$

The total utility of an attribute is the highest (m^2) when the attribute occurs in every class description, and has a different value in each of them.

The second criterion prefers attributes that have fewer values in the rules, because nodes that are assigned such attributes will have a smaller fan out (in the case of continuous attributes, they are quantized, and their new "values" are ranges of original values). The third criterion prefers the attribute that occurs in a larger number of rules, because this can help to evaluate all the rules faster.

These three criteria are combined into one attribute ranking measure using the "lexicographical evaluation functional with tolerances" (LEF; Michalski, 1973). First, the method evaluates attributes on the basis of their utility. If two or more attributes share the same top score or their scores are within the assumed *tolerance range*, the method evaluates these attributes using the second criterion (other attributes are ignored). If again two or more attributes share the same top score, or their scores are within the tolerance range, then the third criterion is used. If there is still a tie, the method selects the "best" attribute randomly.

The second problem of forming decision trees from decision rules is how to start from a single root and represent all the rules in a decision tree, even in cases when some rules do not contain common attributes. One way of overcoming with this problem is to create additional nodes corresponding to "missing" attributes. This method can significantly increase the size of the tree. Another approach, adopted in AQDT-1, is to create a "Don't care" value for attributes that may not be necessary to evaluate.

The following simple example illustrates the AQDT-1 method. Suppose there are three decision classes, C1,C2 & C3, described by the following AQ15-derived DNF expressions:

C1	<=	[x1=3]&[x2=2]	v	[x1=3] &[x3=1 v 3] &[x4=1]
C2	<=	[x1=1 v 2]&[x2=3 v 4]	v	[x1=2] &[x3=1 v 2] &[x4=2]
C3	<=	[x1=1]&[x2=1]	v	[x1=4] &[x3=2 v 3] &[x4=3]

The method turns such descriptions into elementary rules, which have only one attribute value in each condition (no internal disjunction). This is done by "multiplying out" each conjunction in the DNF expressions.

Figure 1 illustrates the process of selecting an attribute for a node in the tree based on the set of elementary rules. The rows "Values in Ci" list values of the attribute in the rules of the decision class Ci. "Value sharing" indicates whether the value of that attribute in the rule of a given class is or is not present in rules of other classes. Attributes x2 and x4 both have total utilities of 9. Therefore, they are evaluated on the second criterion (the number of attribute values in rules). Attribute x4 has fewer values (3) than x2 (4), therefore it is assigned to the root of the tree. In the next step, AQDT-1 modifies its data structure to eliminate the rules containing x4 from all the concept descriptions, and marks x4 as an "in_tree" attribute. This process is repeated until all

rules are eliminated. In this example, there will be two iterations. The second attribute to be chosen is x2 and a don't care value will be added to x4 to merge the tree.

Concept (# of rules)	Attributes				
		x1	x2	x3	x4
C1 (3 rules)	Values in C1	3	2	1, 3	1
	Val. sharing	No	No	Yes	No
	Class utility	3	3	1	3
C2 (6 rules)	Values in C2	1, 2	3 v 4	1 v 2	2
	Val. sharing	Yes	No	Yes	No
	Class utility	2	3	1	3
C3 (3 rules)	Values in C3	1,4	1	2 v 3	3
	Val. sharing	Yes	No	Yes	No
	Class utility	2	3	1	3
Total Attribute Utility		7	9	3	9

Figure 1: Determining the total autility of the attributes.

3. Analyzing Decision Trees Obtained by AQDT-1

The performance of the AQDT-1 method was evaluated by applying it to several learning problems. The most complex problem was to learn a decision rules for characterizing the voting records in the US Congress. Each voting record was described in terms of 19 multivalued attributes. AQDT-1, when run in the conventional mode mode (no "skip-nodes") and with the attribute selection criterion minimizing the number of nodes, produced a decision tree with 20 nodes, and 91.8% predictive accuracy on the testing examples. For comparison, the well-known decision tree learning program, C4.5, was also run on the exactly the same data. C4.5 produced a decision tree with 23 nodes, and 85.7% predictive accuracy. (Both programs were assumed to produce a complete and consistent decision tree with regard to the training examples, i.e., a decision tree that gives 100% correct recognition of the training examples). AQDT-1 was also run in the "skip-node" mode. As shown in Figure 2, the obtained "skip-node" decision tree has only 13 nodes. If the value of "Food_stamp_cap" is 0 or 1 then one can make the decision "Democrat" or "Republican", respectively. The branch "Don't care" stemming from "Food_stamp_cap" allows one to proceed to the next node (i.e., evaluate the next attribute "Occupation") without knowing the value of "Food_stamp_cap".

The idea of "Don't care" branches (or "skip nodes") allows one to build a decision tree from rules that do not intersect logically. Introducing such "Don't care" branches not only makes the resulting "skip-node" decision tree simpler, but also may allow one to reach a decision when values of some attributes are unknown. This way, a "skip-node"

decision tree makes possible in some situations to avoid measuring an attribute when it is not logically necessary, which is a problem with conventional decision trees (Michalski, 1990).

Let us define the *accuracy* of a decision tree against a set of examples as the percentage of examples that are correctly classified by the tree out of the total number of examples. We will distinguish between two types of accuracy: the *level* accuracy and the *accumulative* accuracy. The level accuracy of a leaf node is the percentage of the correctly classified examples at this node. The level accuracy at a non-leaf node is the percentage of the number of examples that are not classified to any class at that node. The accumulative accuracy of a leaf is the *total* percentage of the correctly classified examples at this point, including those classified correctly at higher nodes.

In Figure 2, L denotes the level accuracy, and A denotes the accumulative accuracy at a given node. For example, the leaf node "Republican" at the first level of the tree has level accuracy of 13/20 (65%). The node "Republican" at subsequent levels has the level accuracy 15/20 (75%), 19/20 (95%), and 11/20 (55%), respectively. In contrast, the accumulative accuracy of the "Republican" nodes at different levels is 13/20 (65%), 18/20 (90%), 20/20 (100%), and 20/20 (100%). The accumulative accuracy at the second level was computed by adding the 13 examples classified correctly at level one, and 5 examples classified correctly, out of the 13 unclassified examples at the first level.

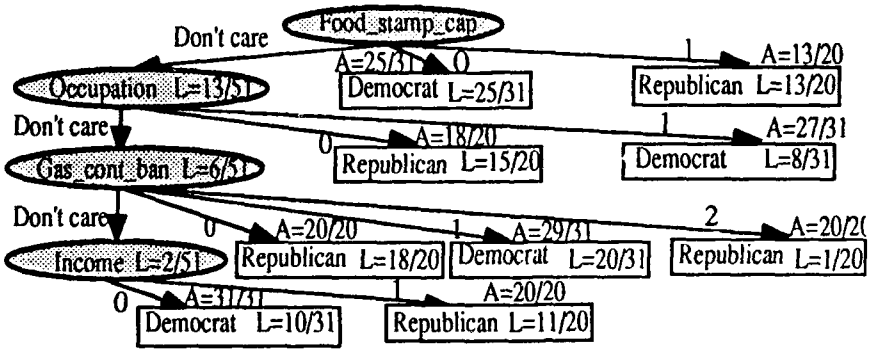


Figure 2: AQDT-1 derived "skip-node" decision tree for the US Congress Voting data.

Let us explain how the AQDT-1-generated tree classifies testing examples. First, it creates an initial hypothetical conclusion by matching the first subtree, which consists of the root and its leaves (in fig. 2), then, it confirms or contradicts this hypothesis through the other subtrees (through Don't care nodes). AQDT-1 uses the level accuracy to choose a conclusion in cases where there is a contradiction with the initial hypothesis. The testing example (Food_stamp_cap=0, Occupation=0, Gas_cont_ban=0, income=2) or (0,0,0,2) will assign an initial conclusion "Democrat" with accuracy 80.6%. During the confirmation process, we notice a contradiction from the second and third subtrees.

The percentages for these subtrees are 75*13/51=19.1% and 95*6/51= 11.1%, respectively. In the last path, there is no value 2, we consider such case against the initial conclusion with percentage (55*2/51=2.2%). If the subtraction (80.6-32.4=48.2%) is less than 50% (in case of two classes), the final conclusion is the contradictory one. Otherwise, and also in cases where there are some subtrees that support the initial conclusion, the following method is used. Assume the example, (0,0,1,1) where the third subtree only supports the initial conclusion. Based on the information in Figure 2, we assume that the level accuracy of each contradictory path is 100%, and we consider the contradictory conclusion as initial one, then compare the contradictory percentage for each conclusion. In our example, consider the level accuracy of "Occupation" as 100%. At that level, there are 15 Republican and 8 Democrat examples are classified correctly. There are 28 (28*100/51=54.9%) unclassified examples. A 54.9% will be the level accuracy of "Food_stamp_cap", and (54.9*25/31=44.7%) is the contradictory percentage to the class Republican. That means if the correct concept is Republican, the contradictory percentage is 44.7% from the first subtree, but if it is Democrat, the contradictory percentage is 19.1% from the second subtree. In this example, the contradictory percentage of the "Income" subtree is also less than 44.7%.

4. Decision Tree Pruning

When input data may have errors (noise), it is often useful to prune the obtained decision tree. Such pruning protects the tree from overfilling. Various approaches have been described in (Mingers, 1989; Breiman, et al, 1984; Quinlan, 1986, 1987; Niblett & Bratko, 1986; Cestnik et al, 1987; Clark & Niblett, 1987; Smyth et al, 1990; Cestnik & Bratko, 1991). These pruning approaches differ in their use of various criteria to decide whether or not to prune at a certain stage.

In this paper, we will study the meaning and effect of pruning on a decision tree learned from rules. Pruning the decision tree will be simpler in terms of where to prune because of the structure of the learned decision tree. The meaning of removing one node of the decision tree is equivalent to removing all the alternative conjunctions (conjunctions which contain the attribute and its value) from the rules. As can be seen, the meaning of pruning here is very crucial because of our assumption that the rule-base is complete (AQ15 guarantees 100% match with the training examples). Our pruning approach takes into consideration that the decision tree is learned from rules, not from examples. The pruning strategy is based on pruning whole nodes at the lowest level with their Don't care node. Figure 3 shows the learned decision tree from the Congressional Voting rules. The pruning takes place at the dotted lines, and the numbers on the left represent the accuracy after exchanging the Don't care node with the associated decision.

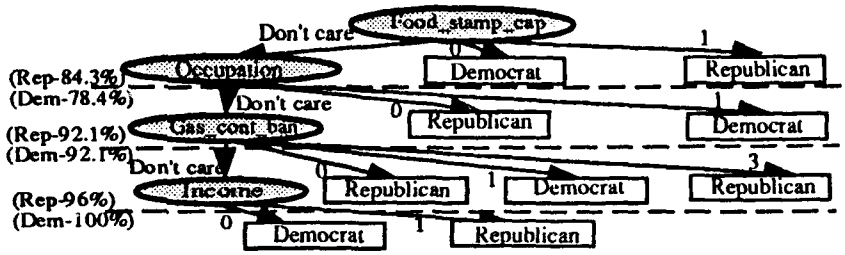


Figure 3: A "skip-node" decision tree and various pruning points at the dotted lines.

5. Comparing Decision Trees from AQDT-1 and C4.5

This section presents an experiment comparing pruned decision trees learned from rules by AQDT-1 with those learned from examples by C4.5. Both programs were applied to the same data on US. congressional voting, described above. The results presented here (for all experiments) are the best results from running C4.5, with both default windows (10 trials; the max. of 20% and twice the square root of the number of examples) and 100% windows (10 trials). C4.5 created a tree with 23 nodes before pruning. After pruning, the tree had 7 nodes with error rate 11.8%.

Figure 4 shows a comparison between the accuracy of the C4.5 and AQDT-1 decision trees with different degrees of pruning. The comparison is done on the training examples (fig. 4a) and testing examples (fig. 4b) examples. From Figure 4, we can see that trees that were learned from rules match the examples better than those trees that were learned from examples with different degrees of pruning.

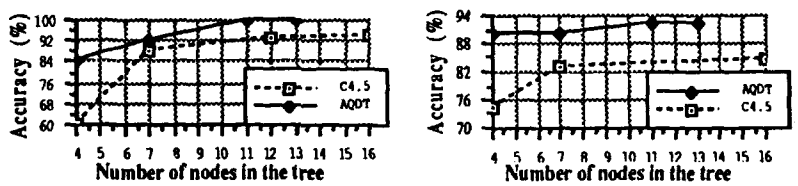


Figure 4: The increase of the accuracy of AQDT-1 and C4.5 generated trees with the number of nodes.

The rest of this section describes two experiments comparing AQDT-1 with C4.5. The experiments were done to show the effect of the size of source data on AQDT-1 and C4.5 in terms of number of nodes and accuracy. In this experiment, the accuracy is calculated against the testing examples. The first experiment involved the second congressional voting data set, which consisted of 112 examples, two concepts to be learned, and 102 testing examples. Figure 5 shows the dependency of accuracy and number of nodes on a set of different relative size of the training examples. The second experiments used the so-called MONK1 data, consists of 124 training examples, two

concepts, and 432 testing examples. Figure 6 shows the dependence of the accuracy and the number of nodes on a set of different relative size of the training examples.

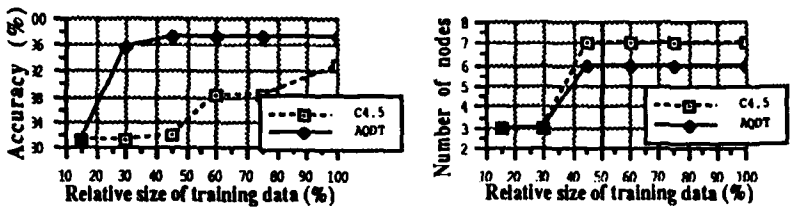


Figure 5 : Comparing decision trees for the US Congressional Voting problem.

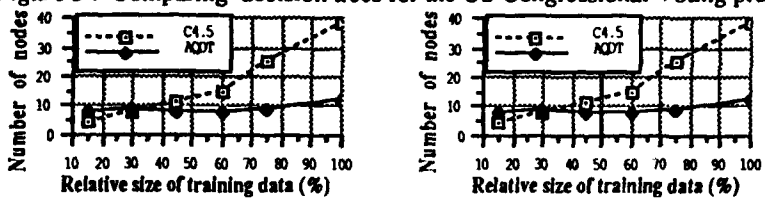


Figure 6: Comparing decision trees for the MONK1 problem.

Figures 5 & 6 show that AQDT-1 produced decision trees with fewer nodes and a higher accuracy than C4.5.

6. Conclusion

The paper presented the AQDT-1 method for efficiently determining decision trees from decision rules. The main difference between determining trees from decision rules and determining them from examples is in the attribute selection function. In the former case, the attribute selection criterion evaluates the role of attributes in the rules, while in the latter it evaluates the coverage of traninig examples. The primary property used in the proposed method is the "total utility" of an attribute for reducing the decision rules.

The method employs the AQ15 inductive learning program for learning decision rules from examples. A major advantage of the proposed approach is that it assists in determining decision trees suitable for different decision making situations, for example, when one wants to avoid measuring some "expensive" attribute. Another advantage is that the method can build decision trees with "Don't care values" on some branches. It has been demonstrated that decision trees with such "Don't care values" can be significantly simpler than conventional decision trees. In the experiments, the AQDT-1 method outperformed the C4.5 method for learning decision trees from examples, both in terms of the predictive accuracy and the simplicity of the generated decision trees.

ACKNOWLEDGMENTS

The authors thank M. Hieb, K. Kaufman, J. Wnek, M. Maloof, H. Vafaie and E. Bloedorn for valuable comments and suggestions.

This research was supported in part by the National Science Foundation under grant No. IRI-9020266, in part by the Defense Advanced Research Projects Agency under the grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research grant No. N00014-91-J-1351.

REFERENCES

Bratko, I. & Lavrac, N. (Eds.), *Progress in Machine Learning*, Sigma Wilmslow, England, Press, 1987.

Bratko, I. & Kononenko, L. "Learning Diagnostic Rules from Incomplete and Noisy Data," *Interactions in AI and statistics*, B. Phelps, (edt.), Gower Technical Press, 1987

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J., "Classification and Regression Trees,",Belmont, California: Wadsworth Int. Group, 1984.

Cestnik, B. & Karalic, A., "The Estimation of Probabilities in Attribute Selection Measures for Decision Tree Induction", *Proceeding of the European Summer School on Machine Learning*, July 22-31, Priory Corsendonk, Belgium, 1991.

Clark, P. & Niblett, T. "Induction in Noisy Domains," *Progress in Machine Learning*, I. Bratko and N. Lavrac, (Eds.), Sigma Press, Wilmslow, 1987.

Hunt, E., Marin, J. & Stone, P., *Experiments in induction*, NY: Academic Press, 1966.

Michalski, R.S. "AQVAL/1-Computer Implementation of a Variable-Valued Logic System VL1 and Examples of its Application to Pattern Recognition," *Proceeding of the First International Joint Conference on Pattern Recognition*, pp. 3-17, 1973.

Michalski, R.S., Mozetic, I., Hong, J. & Lavrac, N., The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains,"*Proceedings of AAAI-86*, Philadelphia, PA, 1986.

Michalski, R.S., "Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-tiered Representation,"*Machine Learning: An Artificial Intelligence Approach, Vol. III*, Y.Kodratoff & R.S.Michalski (Eds.), Morgan Kaufmann, pp. 63-111, 1990.

Mingers, J., "An Empirical Comparison of selection Measures for Decision-Tree Induction," *Machine Learning*, pp.319-342, Vol. 3, No.4, Kluwer Academic Pub., 1989.

Niblett, T. & Bratko, I., "Learning Decision Rules in Noisy Domains," *Proceeding Expert Systems 86*, Brighton, Cambridge: Cambridge University Press, 1986.

Quinlan, J.R., "Discovering Rules By Induction from Large Collections of Examples," *Expert Systems in the Microelectronic Age*, Ed. D. Michie, Edinburgh Unv. Press, 1979.

Quinlan, J.R., "Learning Efficient Classification Procedures and Their Application to Chess End Games," R.S. Michalski, J.G. Carbonell and T.M. Mitchell, (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Los Altos: Morgan Kaufmann, 1983.

Quinlan, J.R., "Induction of Decision Trees,"*Machine Learning* Vol. 1, No. 1, Kluwer Academic Publishers, 1986.

Smyth, P., Goodman, R.M. & Higgins, C., "A Hybrid Rule-based/Bayesian Classifier," *Proceedings of ECAI 90*, Stockholm, August, 1990.

Integrating Machine-Learning Techniques in Knowledge-Based Systems Verification

Hakim Lounis
Laboratoire de Recherche en Informatique
Université de Paris-Sud, Bâtiment 490
91405 Orsay Cedex, FRANCE
email : lounis@lri.lri.fr, tel : (33) 69 41 64 09

Abstract. A significant problem in the development of Knowledge-Based Systems (KBS) is its verification step. This paper describes an expert system verification approach that considers system specifications, and consequently, Knowledge Bases (KB) to be partially described when development starts. This partial description is not necessarily perfect and our work aims at using Machine Learning techniques to progressively improve the quality of expert system Knowledge Bases, by coping with two major KB anomalies : incompleteness and incorrectness. In agreement with the current tendency, KBs considered in our approach are expressed in different formalisms. Results obtained with two different learning algorithms, confirm the hypothesis that integrating machine learning techniques in the verification step of a Knowledge-Based System life cycle, is a promising approach.

Keywords. Verification, Formal Specifications, Machine Learning, Revision Process, Production Rules, Semantic-Net, Integrity Constraint.

1. Introduction

Nowadays there still are few commercialized Knowledge-Based Systems (KBS). The major reason is the lack of a strict validation step in their life cycle. There is a widespread agreement that KBSs cannot be designed in a linear fashion. This is due to the typical problems they have to resolve; they require then to adapt traditional techniques of software development or to use new techniques relevant to Artificial Intelligence (AI) systems. For instance, the life cycle of Figure 1-a does not seem to be advisable for KBS development. The model shown in Figure 1-b allows the developer to partially describe KBS specifications; These specifications will be progressively completed through each new cycle.

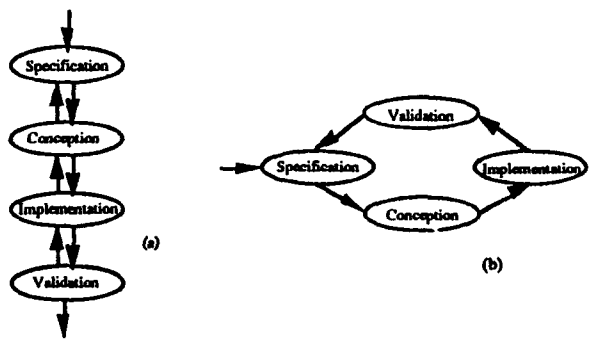


Figure 1 : Different life cycles

To avoid confusion due to lack of unified terminology, we adopt in this paper Laurent's terminology [1] for validation process. We use different concepts for validation purposes. Some are formalizable (e.g., circularity of a rule base, redundancy of a rule base, etc), and some are not (e.g., level of performances, explanation capabilities, etc). From those concepts we may set up specifications, always in a formal way. But we obtain either really formal specifications (with the

formalizable concepts) or what [1] calls pseudo-formal specifications. For example, to deal with explanation capabilities, we can *define a formal process*: e.g., an ad-hoc questionnaire will be filled by ten users; each answer will give points, and a formal process of aggregation will produce a final note N in a given scale (e.g., [0 .. 100]). If we choose a threshold, e.g., 80, then we can set up a pseudo-formal specification: $N \geq 80$. This leads to the following definitions:

Definition 1 : A validation process is a process that attempts to determine whether a KBS satisfies or not one of its specifications.

Definition 2 : A verification process is a validation process that attempts to determine whether a KBS satisfies or not one of its formal specifications.

Definition 3 : An evaluation process is a validation process that attempts to determine whether a KBS satisfies or not one of its pseudo-formal specifications.

We can retain that the most fundamental difference between a verification process and an evaluation process concerns the interpretation of the result: it is always possible to conclude whether the KBS fits or not the formal specification, as well as the corresponding informal specification in natural language. However, this is not the case with an evaluation process because we cannot conclude without ambiguity whether the KBS satisfies or not the informal specification that led to the pseudo-formal specification.

This paper presents an expert system verification approach that considers system specifications to be partially described when development starts. This partial description is not necessarily perfect and our work aims at using Machine Learning techniques to progressively improve the quality of expert system Knowledge Bases (KB), by coping with two major KB anomalies: incompleteness and incorrectness. By integrating Machine Learning techniques in the validation step of a KBS evolutionary life cycle model (e.g., Figure 1-b), we allow experts to propose an initial version of the KB which will be refined and corrected throughout a refinement cycle. This particular point permits to deal with a drawback of pure inductive learning algorithms (e.g., ID3 [2], AQ [3], ...): induced rules are often not directly usable by current expert system's shell. The reason is that such rules are generally flat and so do not permit the KBS to have explanation capabilities. We believe that starting with an initial KB, possibly imperfect, which will be refined until it reaches its final expression, is a better way than that using directly pure inductive learning. Figure 2 presents a view similar to that previously presented, except that here we deal with life cycle of a KB:

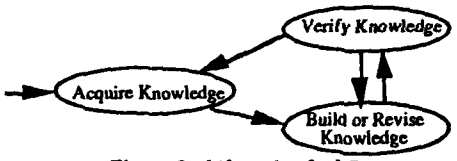


Figure 2 : Life cycle of a KB

In our approach, we consider expert systems that deal with KBs expressed in different formalisms (This is in accordance with the current tendency). Each part of the knowledge is represented in a particular formalism that is considered as the most appropriate, relatively to the type of knowledge it expresses. The knowledge we consider consists of three parts : shallow knowledge, a deeper kind of knowledge, and a set of examples. The first part is a set of production rules expressed in First Order Logic (FOL); the second part consists in turn, of two categories of knowledge: Semantic nets representing entities of the application domain and their relationships, and a set of integrity constraints. Lastly, the set of examples contains observations represented as conjunctions of first order literals. Each observation is classified as belonging to a given concept.

In this paper, we first describe previous works in the field of KBS verification and then compare them to our method and we highlight the strengths and weakness of each approach. We then present our approach and illustrate it with an example. In this example, in order to revise the KB, we have integrated in the verification cycle two different learning algorithms, FOIL [4] and KBG [5].

2. Related works

Earlier work in the field of verification of KB have not integrated machine learning techniques to revise initial formulation of KBs. They generally consider rule bases expressed in attribute-value logic or first-order logic. CHECK [6] verifies statically consistency and completeness of KBs expressed in FOL. This method is typical of the approaches that are easy to implement. Despite of the simplicity of the basic concepts used in CHECK, a rule base may present incoherences that CHECK could not detect.

To solve such weakness, new approaches referred to as dynamic, have been evolved. These ones take into account the deductive power of knowledge bases. These methods are either exhaustive, in which case they aim at finding the specification of all incoherent situations, or heuristic and so, they exploit heuristics to select the more "interesting" conjectures of incoherence. The exhaustive approach is used by systems like COVADIS [7] and COCO [8]. Starting with incoherence specification, the issue is to prove that these specifications are reachable from sets of "incoherent facts". If this is the case, the KB is incoherent; otherwise it is coherent. A typical example of systems adopting a heuristical approach, is the SACCO [9] system. This approach brings computation speed-up, and avoids proposing to the expert real but improbable conjectures. To do this, SACCO makes use of heuristics that allow to define a limited set of potential conjectures of incoherence. If the so defined conjectures are verified by an initial fact base, then the incoherence is detected; otherwise, the KB is assumed coherent.

On the other hand, functional verification of KBS makes sure that the provided results are in accordance with domain's semantic. For instance, from an expert knowledge and a set of cases, the SEEK [10] system exploits a rule refinement cycle, which by performance evaluation of the rules on a library of cases, and by analyzing the statistical behavior of each rule, suggests modifications to be introduced in the expert knowledge.

More recently, several works have integrated machine learning algorithms in order to revise automatically imperfect KBs. They generally treat KBs expressed in the form of production rules. Some systems are only capable of generalizing an overly specific (incomplete) KB [11, 12, 13] while others are only capable of specializing an overly general KB [14, 15, 16].

Number of these systems, uses the Explanation-Based Learning (EBL) [17] approach to deal with imperfect KBs. For instance, to deal with overly general KBs, EBL/TS [18] uses the explanation tree of each positive example and then replace overly general definition of the given concept, by the rule associated with this explanation tree. On the other hand, A-EBL [18] treats the problem of multiple example explanations. It throw out inconsistent explanations and retain only a minimal set of "good" explanations. This is done by using heuristics. Such approaches starts with an imperfect KB expressed in terms of rules, and replace the entire KB by the rule associated to the explanation tree, revising the operational definition of a given concept. These processes do not preserve the structural form of rule bases and produce generally flat rules.

However, the current tendency in knowledge representation aims at integrating different formalisms such rules, frames, semantic-nets, etc. The goal pursued by such an approach is the increasing of explanation capabilities of KBSs and acquisition of different kind of knowledge, each expressed in a particular formalism, considered as the most appropriate, relatively to the type of knowledge it expresses. In the field of Knowledge Verification, there still is few systems that cope with imperfect KBs, expressed in different formalisms. Our approach considers KBs expressed in different formalisms as : rule bases, semantic-nets and integrity constraints. It considers that using machine learning techniques, in an evolutionary life-cycle can propose refinements to the initial KB, until it reaches a correct and complete expression. In this way, we deal with a drawback of pure inductive learning algorithms: induced rules are often not directly usable by expert system's shells.

3. Description of the approach

Our method starts with the hypothesis that the initial KB provided by an expert is probably incomplete and/or incorrect. The detection of incompleteness and/or incorrectness is centred on the notion of label of a given concept.

Definition 4 : The label of a concept is the set of all initial[1] facts (i.e., literals) that allows a KBS to deduce[2] the concept :

$E_{concept} = V_{i=1,n} \Lambda_{j=1,m} P_{ij}$, where P_{ij} is an initial first order literal.

The Knowledge provided to the system includes:

- Rules of expertise: $\{R_i / R_i = (\Lambda_{k=1,\ m} P_{ik}) \Rightarrow Q_i \}$ where P_{ik} and Q_i are first order literals.
- A semantic net describing domain entities and relations between entities.
- Integrity constraints: $\{I_k / I_k$ = incompatibility (concept-i, concept-j)$\}$ where concept-i and concept-j are first order literals.
- A set of examples : $\{e / e = desc(e) \Lambda class(e)\}$ where $desc(e) = \Lambda_{j=1,\ p} L_j$ is the description of the example; this description is a conjunction of initial first order literals. class(e) is a first order literal that indicates the concept to which the example belongs.

All literals are typed. This means that arguments of a given predicate takes there values in a user defined type. For instance, we have considered the following types: nominal, linear, integer, real, hierarchical.

In this context, an incorrectness corresponds to the case where an example belongs to a concept that is incompatible with its real concept. Such an incompatibility is stated by the meta-predicate *incompatible (concept-i, concept-j)*. According to our definition, an incorrectness is detected if the following formal specification is verified:

(A) $\exists$ *e: desc(e)* $= \Lambda_{j=1,\ p} L_j$ *&* $class(e) = concept\text{-}i$, $\exists$ *an integrity constraint I = incompatible (concept-i, concept-j),* $\exists$ *a substitution* σ *such that* $desc(e) \Rightarrow \{E_{concept\text{-}j}\}\sigma$.

On the other hand, an incompleteness corresponds to the case where an example does not belong to its real concept label. The formal specification associated to the latter informal one is the following:

(B) $\exists$ *e: desc(e)* $= \Lambda_{j=1,\ p} L_j$ *&* $class(e) = concept\text{-}i$, *such that there is no substitution* σ *so that that* $desc(e) \Rightarrow \{E_{concept\text{-}i}\}\sigma$.

As stated by [19], imperfections of the KB are due to many reasons. In our case, when an incompleteness is detected, the revision process has to deal with many cases. It can induce a new rule which has to be linked up in an appropriate place within the structural representation of rules. It can also, generalize an existing rule by dropping literal(s) in its left-hand side or by learning a more general literal. In the case of incorrectness, the revision process aims at specializing the label. This is firstly done by the localization of the faulty rule(s) and subsequently by specializing their left-hand side. This specialization is done by learning new literals or by specializing an existing one. A rule can also be suppressed if it is satisfied exclusively by counter-examples of the studied concept.

Unlike many other approaches, the revision process concerns a faulty sub-concept. A sub-concept is defined by a non-initial literal that appears in the definition of the studied concept. Refinements proposed by the learning tool are performed thanks to a learning algorithm, which considers as input data, a set of examples depending on the kind of detected anomalies (incorrectness or incompleteness). In this way, the learned process concerns only the label of this sub-concept, without modifying the label of other sub-concepts. Therefore, they allow us to preserve the structural form of the rule base (i.e., the shallow knowledge). In the case of a detected incorrectness, the revision algorithm works with positive examples of the given concept and negative ones that are subsumed by the faulty concept label. In the case of incompleteness, the considered positive examples are those that verify the formal specification (B) and do not verify the actual definition of the faulty sub-concept. The negative ones are those subsumed by the current sub-concept definition, and those imposed by integrity constraints. We

[1] Initial facts are used to describ examples. This notion is close to the notion of operationality introduced by in EBL.

[2] The considered inference engine strategy is simple: a rule is fireable if its left-hand side is verified. If more than one rule is fireable at a moment, the inference engine considers the first in the KB. All fireable rules are considered.

must notice that for each example in the learning-set, a new description called "contextual description" is proposed. It contains literals that have a semantic closeness with the faulty concept. They are suggested by the semantic-net, or by a domain expert. The intervention of an expert may be necessary if it turns out that the semantic-net is incomplete.

This revision process may propose some modifications in the deep knowledge base. Such modifications arise when the initial vocabulary is not sufficient; in which case, the semantic net has to be extended with an entity or a relation. These latter correspond respectively to a predicate argument or a predicate. The growing of the initial vocabulary is made easier by the fact that the semantic net is organized in different pieces, each piece concerning a particular sub-concept (e.g., in the mammal example this will correspond to particular functionalities).

We can summarize our verification algorithm as follows:

a) Determine the label E_C of a target concept.
b) Determine all examples that are subsumed by the current label E_C.
c) Identify faulty concepts.
d) For each faulty concept,
   - Build the leaning set;
   - For each example in the learning set, propose a contextual description;
   - Learn a new piece of knowledge;
   - Revise the KB.

Before illustration of our approach on a particular example, we summarize the entire revision process by Figure 5 :

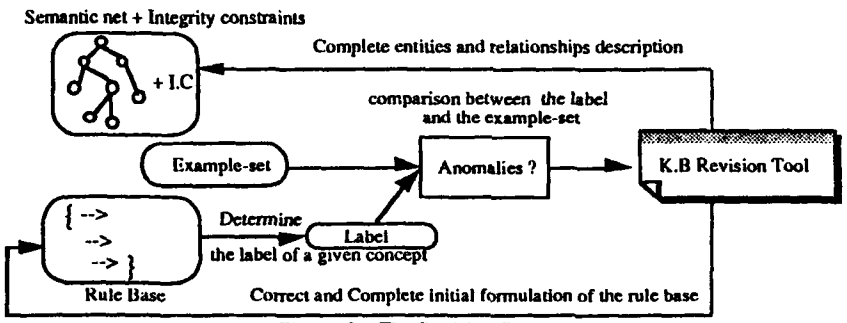


Figure 3 : The Revision Process

4. An example

The Knowledge Base presented in this section is drawn from [20]. It is made up of different pieces of knowledge, each piece being represented in a particular formalism which is considered as the most appropriate, relatively to the knowledge it expresses. First, a rule base expressed in First Order Logic, that contain expert's opinion in the mammal theory. The second piece of knowledge is a semantic-net describing domain entities and their relationships. It is a deeper kind of knowledge than expert's opinion. Figure 7 shows these two kinds of knowledge:

mammal(x) <-- blood-system(x, mammal) Λ sexual-life(x, mammal) Λ locomotion(x, mammal)
blood-system(x, mammal) <-- blood(x, y) Λ temperature(y, hot) Λ heart(x, z) Λ chambers(z, 4)
sexual-life(x, mammal) <-- fertilization(x, internal) Λ way-of-develop(x, mammal)
locomotion(x, mammal) <-- ante-limbs(x, legs) Λ post-limbs(x, legs) Λ move(x, ground)
way-of-develop(x, mammal) <-- develop(x, placenta) Λ reproduction(x, viviparous) Λ has(x, breasts)
bird(x) <-- reproduction(x, oviparous) Λ blood(x, y) Λ temperature(y, hot) Λ mouth (x, bill)

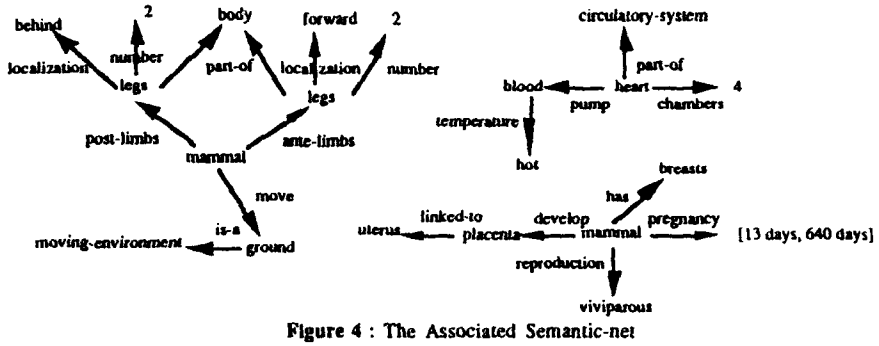


Figure 4 : The Associated Semantic-net

In addition, integrity constraints are expressed in the following way :

incompatibility (mammal (x), bird (x))

incompatibility (way-of-develop (x, mammal), way-of-develop (x, bird))

Finally , we have a set of classified examples of mammals and birds. Some of them are listed in the following table:

cat : mammal(cat) Λ blood(cat, b) Λ temperature(b, hot) Λ fertilization(cat, internal) Λ develop(cat, placenta) Λ reproduction(cat, viviparous) Λ has(cat, breasts) Λ ante-limbs(cat, legs) Λ post-limbs(cat, legs) Λ move(cat, ground) Λ heart(cat, h) Λ chambers(h, 4).

whale : mammal(whale) Λ blood(whale, b) Λ temperature(b, hot) Λ fertilization(whale, internal) Λ develop(whale, placenta) Λ reproduction(whale, viviparous) Λ has(whale, breasts) Λ ante-limbs(whale, fines) Λ post-limbs(whale, tail) Λ move(whale, water) Λ heart(whale, h) Λ chambers(h, 4) Λ size(whale, giant).

dolphin : mammal(dolphin) Λ blood(dolphin, b) Λ temperature(b, hot) Λ fertilization(dolphin, internal) Λ develop(dolphin, placenta) Λ reproduction(dolphin, viviparous) Λ has(dolphin, breasts) Λ ante-limbs(dolphin, fines) Λ post-limbs(dolphin, tail) Λ move(dolphin, water) Λ heart(dolphin, h) Λ chambers(h, 4) Λ behaviour(dolphin, friendly).

kangaroo : mammal(kangaroo) Λ blood(kangaroo, b) Λ temperature(b, hot) Λ fertilization(kangaroo, internal) Λ develop(kangaroo, marsupium) Λ reproduction(kangaroo, viviparous) Λ has(kangaroo, breasts) Λ ante-limbs(kangaroo, legs) Λ post-limbs(kangaroo, legs) Λ move(kangaroo, ground) Λ heart(kangaroo, h) Λ chambers(h, 4).

bat : mammal(bat) Λ blood(bat, b) Λ temperature(b, hot) Λ fertilization(bat, internal) Λ develop(bat, marsupium) Λ reproduction(bat, viviparous) Λ has(bat, breasts) Λ ante-limbs(bat, wings) Λ post-limbs(bat, legs) Λ move(bat, air) Λ heart(bat, h) Λ chambers(h, 4).

spiny-anteater : mammal(spiny-anteater) Λ blood(spiny-anteater, b) Λ temperature(b, hot) Λ fertilization(spiny-anteater, internal) Λ mouth(spiny-anteater, bill) Λ reproduction(spiny-anteater, oviparous) Λ has(spiny-anteater, breasts) Λ ante-limbs(spiny-anteater, legs) Λ post-limbs(spiny-anteater, legs) Λ move(spiny-anteater, ground) Λ heart(spiny-anteater, h) Λ chambers(h, 4).

ornythorynchus : mammal(ornithorynchus) Λ blood(ornythorynchus, b) Λ temperature(b, hot) Λ fertilization(ornythorynchus, internal) Λ mouth(ornythorynchus, bill) Λ reproduction(ornythorynchus, oviparous) Λ has(ornythorynchus, breasts) Λ ante-limbs(ornythorynchus, legs) Λ post-limbs(ornythorynchus, legs) Λ move(ornythorynchus, ground) Λ heart(ornythorynchus, h) Λ chambers(h, 4).

duck : bird(duck) Λ blood(duck, b) Λ temperature(b, hot) Λ fertilization(duck, internal) Λ mouth(duck, bill) Λ reproduction(duck, oviparous) Λ covered(duck, feathers) Λ ante-limbs(duck, wings) Λ post-limbs(duck, legs) Λ move(duck, air) Λ heart(duck, h) Λ chambers(h, 4).

crow : bird(crow) Λ blood(crow, b) Λ temperature(b, hot) Λ fertilization(crow, internal) Λ mouth(crow, bill) Λ reproduction(crow, oviparous) Λ covered(crow, feathers) Λ color(feathers, black) Λ ante-limbs(crow, wings) Λ post-limbs(crow, legs) Λ move(crow, air) Λ heart(crow, h) Λ chambers(h, 4).

Table 1 : Examples of mammals and birds

As mentioned above, our approach starts by determining concept's label. For instance, if we want to study the initial mammal's definition, we obtain the following label:

E_{mammal} = *blood(x, y) Λ temperature(y, hot) Λ heart(x, z) Λ chambers(z, 4) Λ fertilization(x, internal) Λ develop(x, placenta) Λ reproduction(x, viviparous) Λ has(x, breasts) Λ ante-limbs(x, legs) Λ post-limbs(x, legs) Λ move(x, ground)*

We can notice that many mammal examples are not covered by this current label of mammal concept. For instance, $\exists e$: *class(e) = mammal / there is no substitution* σ *so that: desc(e)*$\Rightarrow${E_{mammal}}σ. This is the case for e $\in$ {whale, dolphin, kangaroo, bat, spiny-anteater, ornithorynchus}. On the other hand, we notice that examples of mammal, that is a concept incompatible with bird's concept, are covered by the present bird label, i.e., $\exists e$ *: class(e) = concept-j and incompatible (concept-j, bird) / there is a substitution* σ *so that desc(e)*$\Rightarrow$ {E_{bird}}σ. The concerned examples are the following: {spiny-anteater, ornithorynchus}. In such a case, both incompleteness and incorrectness are detected.

To learn new definitions of faulty concepts, we have used two different learning algorithms : FOIL and KBG. FOIL learn Horn clause from examples of relations (in our case, relations are the predicates of the domain application). The target relation is the faulty concept to revise. Like ID3, this algorithm employs an information-gain estimate to select the best literal. On the other hand, KBG is a learning algorithm which uses a similarity measure to compute a distance between the different entities of a pair of examples.

To deal with these anomalies, the revision process starts by identifying faulty concepts. For instance, It finds that both locomotion(x, mammal) and way-of-develop(x, mammal) are sub-concepts which need to be completed. To complete the locomotion definition, the revision process build a learning set, containing as positive examples, all mammal examples that are not covered by the current locomotion definition, and as negative ones, those that are covered by this definition and those that are imposed by integrity constraints. We obtain the following learning set:

POS={dolphin, whale, bat}
NEG={cat, kangaroo, spiny-anteater, ornythorynchus}

For each of these examples, our revision process builds a contextual description. It contains literals which have a semantic closeness with the faulty concept. For the previous learning set, we obtain these descriptions:

$\mathcal{D}$*{dolphin)=ante-limbs(dolphin, fines) Λ post-limbs(dolphin, tail) Λ move(dolphin, water)*

$\mathcal{D}$*{whale)=ante-limbs(whale, fines) Λ post-limbs(whale, tail) Λ move(whale, water)*

$\mathcal{D}$*{bat)=ante-limbs(bat, wings) Λ post-limbs(bat, legs) Λ move(bat, air).*

$\mathcal{D}$*{cat)=ante-limbs(cat, legs) Λ post-limbs(cat, legs) Λ move(cat, ground)*

$\mathcal{D}$*{kangaroo)=ante-limbs(kangaroo, legs) Λ post-limbs(kangaroo, legs) Λ move(kangaroo, ground)*

$\mathcal{D}$*{spiny-anteater)=ante-limbs(spiny-anteater, legs) Λ post-limbs(spiny-anteater, legs) Λ move(spiny-anteater, ground)*

$\mathcal{D}$*{ornythorynchus)=ante-limbs(ornythorynchus, legs) Λ post-limbs(ornythorynchus, legs) Λ move(ornythorynchus, ground).*

From both FOIL and KBG, we obtain the following new piece of knowledge :
locomotion(x, mammal) <-- ante-limbs(x, wings) V ante-limbs(x, fines).

This new expression complete the mammal label, and precisely the locomotion one. We obtain then the following new definition:
locomotion(x, mammal) <-- [ante-limbs(x, legs) Λ post-limbs(x, legs) Λ move(x, ground)] V ante-limbs(x, wings) V ante-limbs(x, fines).

In the same time, the learning algorithm completes the "locomotion" piece of the semantic-net concerned with previous modifications. It has to extend its initial vocabulary:

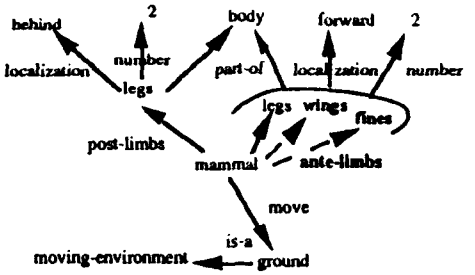


Figure 5 : Completion of the "locomotion" piece in the semantic-net

To complete the solution of incompleteness problem, the revision process considers as positive examples, the following examples: *kangaroo, bat, spiny-anteater, and ornithorynchus*, that are not covered by the current definition of *way-of-develop(x, mammal)*. Negative ones are: *cat, whale, dolphin, duck and crow*. We notice here that *duck* and *crow* are added in the set of counter examples, because of the integrity constraint : *incompatibility(way-of-develop(x, mammal), way-of-develop(x, bird))*. In this case, we obtain the following contextual descriptions:

$\mathcal{D}_{\mathcal{C}}$(kangaroo)=reproduction(kangaroo, viviparous) Λ has(kangaroo, breasts) Λ develop(kangaroo, marsupium).

$\mathcal{D}_{\mathcal{C}}$(bat)=reproduction(bat, viviparous) Λ has(bat, breasts) Λ develop(bat, marsupium).

$\mathcal{D}_{\mathcal{C}}$(spiny-anteater)=reproduction(spiny-anteater, oviparous) Λ has(spiny-anteater, breasts).

$\mathcal{D}_{\mathcal{C}}$(ornythorynchus)=reproduction(ornythorynchus, oviparous) Λ has(ornythorynchus, breasts).

$\mathcal{D}_{\mathcal{C}}$(cat)=reproduction(cat, viviparous) Λ has(cat, breasts) Λ develop(cat, placenta).

$\mathcal{D}_{\mathcal{C}}$(dolphin)=reproduction(dolphin, viviparous) Λ has(dolphin, breasts) Λ develop(dolphin, placenta).

$\mathcal{D}_{\mathcal{C}}$(whale)=reproduction(whale, viviparous) Λ has(whale, breasts) Λ develop(whale, placenta).

$\mathcal{D}_{\mathcal{C}}$(duck)=reproduction(duck, oviparous) Λ covered(duck, feathers).

$\mathcal{D}_{\mathcal{C}}$(crow)=reproduction(crow, oviparous) Λ covered(crow, feathers).

From this learning set, FOIL induces an other definition of way-of-develop(x, mammal): *[reproduction (x, oviparous) Λ has (x, breasts)] V develop(x, marsupium)*. On the other hand, KBG learns the following definition: *[reproduction (x, oviparous) Λ has (x, breasts)] V [develop(x, marsupium) Λ has(x, breasts) Λ reproduction(x, viviparous)*.

Finally, the revision process completes the previous *way-of-develop(x, mammal)* definition as follows:

way-of-develop (x, mammal) <-- [develop (x, development-organ) Λ reproduction (x, viviparous) Λ has (x, breasts)] V [reproduction (x, oviparous) Λ has (x, breasts)].

As in the previous case, we complete the semantic net portion, concerned with mammal sexual life. We add the fact that both marsupium and placenta are development-organ, and that some mammals are oviparous:

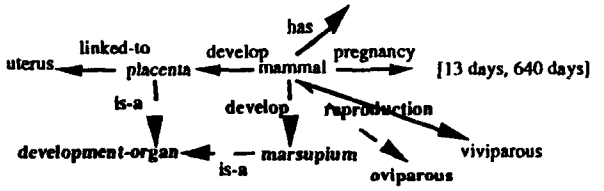


Figure 6 : Completion of the "reproduction" piece in the semantic-net

To solve incorrectness detected thanks to bird's label, the revision process selects all bird examples and the two mammal examples (i.e., *spiny-anteater, and ornithorynchus*), which have help us to detect such incorrectness. We obtain the following learning set:

POS={duck, crow}
NEG={spiny-anteater, ornythorynchus}

$\mathcal{D}_{\mathcal{C}}$*(duck)=covered(duck, feathers)* Λ *ante-limbs(duck, wings)* Λ *move(duck, air).*

$\mathcal{D}_{\mathcal{C}}$*(crow)=covered(crow, feathers)* Λ *ante-limbs(crow, wings)* Λ *move(crow, air).*

$\mathcal{D}_{\mathcal{C}}$*(spiny-anteater)=ante-limbs(spiny-anteater, legs)* Λ *move(spiny-anteater, ground).*

$\mathcal{D}_{\mathcal{C}}$*(ornythorynchus)=ante-limbs(ornythorynchus, legs)* Λ *move(ornythorynchus, ground).*

The literal *"covered (x, feathers)"* is induced by FOIL. KBG learns the literal *"ante-limbs(x, wings)"*.
Finally, the initial bird label is specialized, and we may obtain these two definitions:

bird (x) <-- reproduction (x, oviparous) Λ *blood (x, y)* Λ *temperature (y, hot)* Λ *mouth (x, bill)* Λ *covered (x, feathers).*
or,
bird (x) <-- reproduction (x, oviparous) Λ *blood (x, y)* Λ *temperature (y, hot)* Λ *mouth (x, bill)* Λ *ante-limbs (x, wings).*

This example shows us how our approach deal with incomplete and/or incorrect KB. It considers a kind of knowledge easier to produce by a domain expert, than a set of rules: reliable examples and counter-examples of a given concept.

5. Conclusion

In this research, we address the issue of KB's verification. These KBs consisting of two levels of knowledge differently structured. This paper is centred around the following statement: integration of learning process in a validation step of KBS's life cycle, allows the detection and correction of anomalies present in the initial formulation of the KB.

Our approach exploits a set of examples that experts can provide more easily, than a knowledge directly expressed in the form of rules. Thanks to the notion of concept's label, this approach permits, as a first step, to locate the level at which incompleteness or incorrectness occures. A subsequent step, which makes use of the concerned examples, calls for learning techniques that perform corrections at a level indicated by the incoherent label. In parallel, the description of the deep KB is completed to take account of the new changes. This process permits to start with a initial imperfect KB, and then to correct and complete it until it reaches its final formulation.

Currently, we are exploring the possibility of learning Integrity Constraints. We aims at starting with an Integrity Constraint I_c = Incompatibility (concept-i, concept-j), and then learn from the example set, incompatible environment expressed in terms of initial facts I_c' = incompatibility ($\Lambda_{i=1,k} P_i$). Such a process could reject initial example description, and then avoid the case where the learning algorithm has to deal with erroneous example's description.

Acknowledgement

I would like to thank Yves Kodratoff, my thesis supervisor, for the support he gave to this work and all the members of the Inference and Learning group at LRI.

References

1. Laurent : "Vers une terminologie valide pour le domaine de la validation", actes des JFVAV, pp 1-15, Dourdan, Avril 1992.

2. Quinlan : "Learning Efficient Classification Procedures and their Application to Chess End Games", in Machine Learning : An Artificial Intelligence Approach, R.S. Michalski, J.G.Carbonell & T.M.Mitchell (Eds.), Morgan Kaufmann 1983, pp 463-482.

3. Michalski : "A Theory and a Methodology of Inductive Learning", in Machine Learning : An Artificial Intelligence Approach, R.S. Michalski, J.G.Carbonell & T.M.Mitchell (Eds.), Morgan Kaufmann 1983, pp 83-134.

4. Quinlan : "Learning Logical Definitions from Relations", in Machine Learning Journal, 5, pp 239-266, 1990.

5. Bisson : "Conceptual Clustering in a First-Order Logic Representation", Proceeding of 10th ECAI, Vienna 1992.

6. Nguyen & al. : "Checking an Expert System Knoweldge Base for consistency and completeness", IJCAI 1985, pp 375-379.

7. Rousset : "On the Consistency of Knowledge Bases : The COVADIS System", ECAI 1988, pp 79-84.

8. Loiseau : "Validation, acquisition et mise au point interactive des BC : le système COCO-X fondé sur la cohérence", thèse de doctorat, univeristé de Paris-Sud, 1990.

9. Ayel : "Détection d'incohérences dans les bases de connaissances : SACCO", thèse d'état, Chambery, 1987.

10. Politakis & al. : "Using Empirical Analysis to Refine E.S Knowledge Bases", Artificial Intelligence 22, pp 23-48, 1984.

11. Wilkins : "Knowledge base refinement using apprenticeship learning techniques. In Proceedings of the 7th National Conference on Artificial Intelligence, pp 646-651, St. Paul, MN, August 1988.

12. Danyluk : "Finding new rules for incomplete theories: explicit biases for induction with contextual information". In proceedings of the 6th International Workshop on Machine Learning, pp 34-36, Ithaca, NY, June 1989.

13. Whitehall : "Knowledge-Based Learning: An Integration of Deductive and Inductive Learning for Knowledge Base Completion", PhD thesis, University of Illinois, Urbana, IL, October 1990.

14. Flann & Dietterich : "A study of explanation-based methods for inductive learning". Machine Learning, 4 (2), pp 187-226, 1989.

15. Mooney & Ourston : "Induction over the unexplained: Integrated learning of concepts with box explainable and conventional aspects". In proceedings of the 6th International Workshop on Machine Learning, pp 5-7, Ithaca, NY, June 1989.

16. Cohen : "Learning from textbook knowledge: A case study. In proceedings of the 8th National Conference on Artificial Intelligence, pp 743-748, Boston, MA, July 1990.

17. Mitchell & al. : "EBL : An Unifying View", ML Journal, vol 1, number 1, Kluwer Academic Publishers, 1986, pp 47-80.

18. Cohen : "Abductive Explanation-Based Learning: A Solution to the multiple Inconsistent Explanation Problem", in Machine Learning Journal, Vol 8, number 2, March 1992.

19. Rajamoney & DeJong : "The Classification, Detection and Handling of Imperfect Theory Problems", IJCAI 1987, pp 205-207.

20. Matwin & Plante : "A Deductive-Inductive Method For Theory Revision", IWML 1991, pp 160-174.

Automatic Theorem Generation in Plane Geometry

R. Bagai, V. Shanbhogue, J. M. Żytkow, S. C. Chou

Department of Computer Science
Wichita State University
Wichita, KS 67260-0083, USA
{*bagai, vasant, zytkow, chou*}*@cs.twsu.edu*

Abstract. We introduce a conceptual framework for discovery of theorems in geometry and a mechanism which systematically discovers such theorems. Our mechanism incrementally generates geometrical situations, makes conjectures about them, uses a geometry theorem prover to determine the consistency of situations, and keeps valid conjectures as theorems. We define geometry situations, situation descriptions, theorems, and their relationships important to understand our discovery task. An exhaustive generator of situation descriptions has enormous combinatorial complexity. We analyze various ways to reduce that complexity. Ideally, the generator should create a single description of each situation, should generate more general situations before more specific ones, and should use the previously discovered theorems to constrain its generation mechanism. We describe our generator which possesses most of these properties, and we outline further improvements. Our theorem prover is based on Wu's algebraic method for proving geometry theorems. We discuss the interface between our situation generator and theorem prover and the limitations of our discovery system. Examples of theorems discovered by our system are also presented.

1 Introduction

For over 2000 years, plane geometry has enjoyed the focus of mathematicians who discovered many enormously complex and elegant theorems, and new results are still being discovered. However, the research strategies employed by humans to arrive at these theorems are far from clear. Moreover, no attempt has been made to develop a mechanism that can systematically discover all, or even a well-defined class of theorems. So far, the only substantial approaches to automated discovery in mathematics were directed at the theories of numbers [4, 6, 9]. As much as geometry differs from theories of numbers, one may expect another discovery mechanism for geometry.

In this paper, we concentrate on automated discovery of theorems in geometry. However, if all statements true in the domain of geometry are understood as theorems, we certainly do not wish to discover explicitly all of them. We only want to discover a necessary minimum of theorems which characterize different geometrical situations. We define the notion of a situation, and the necessary and sufficient set of theorems about situations. This leads to a mechanism which systematically formulates conjectures to cover that set of theorems.

Conjectures must be proved before we consider them theorems. We use an algebraic procedure developed by Wu [10, 11] for proving conjectures in geometry. Each conjecture is considered by our theorem prover and, if proved, becomes a theorem. Wu's technique is suitable for conjectures in which the assumption and the conclusion can be expressed as conjunction of polynomial equations and/or their negations (inequations). Conjectures about situations constructed by our mechanism fall into this category. A prover that incorporates Wu's technique was constructed by Chou [1], and this prover has been used to prove hundreds of conjectures. Many of these conjectures were previously known theorems, but some were genuine guesses. Examples of significant new theorems proved by this prover can be found in [2].

Ideally, the conjecture generator should consider each situation and make all independent conjectures about that situation, but should do it only once. It should also generate simpler situations first, so that each theorem can be detected in the simplest possible situation to which it belongs. This way we would not lose any theorems nor would we repeat any effort. We present a computational mechanism which builds situations from points, lines, and a few relationships among these objects. Other types of objects and relationships could be easily included into that mechanism. We show how the verdict returned by the prover influences further generation of conjectures.

Despite the early lead before 1980, automated discovery in mathematics has been left behind two other areas of machine discovery — *scientific discovery* [5, 8] and discovery in databases [7] — both in the number of people involved and the scope of research. Lacking efficient theorem proving capabilities, discovery systems in mathematics, such as AM [6] and CYRANO [4], only make conjectures based on simple induction. Since efficient theorem provers are now available in geometry [1], we can use them to advance machine discovery in mathematics. In comparison to AM, our system discovers not merely conjectures which have been validated by only a few cases in an infinite domain, but theorems, which have been validated according to the proof standards used in mathematics. Our system does not consider empirical evidence. Complying with the rules of the game in mathematics, it deals with ideal geometry in which situations are described by mathematical statements.

2 Situations and Theorems

Intuitively, situations are "pictures" that can be drawn on the plane using basic geometric objects, such as points, lines, circles etc., arranged in certain relationships among each other. In the subsequent discussion and in our current system, we limit ourselves to points and straight lines, but the method can be generalized easily. Because we concentrate on geometry as a branch of mathematics, not as an empirical science, we do not want to study individual pictures and their empirical properties. Mathematical geometry describes pictures by geometry statements, and draws conclusions from those statements. In such a geometry, a class of pictures satisfying a particular description, rather than an individual picture, is a meaningful entity. We will call such classes *situations*. Situations consist of objects and relationships between them. All objects in a situation are assumed to be distinct.

The situations are described by statements which we will call *situation descriptions*.

Formally, a situation description consists of a set of literals (atomic statements and their negations) in a given vocabulary of functions on objects and relationships among objects. Descriptions (and the corresponding situations) can be built incrementally. Consider a simple incremental way of constructing a situation leading to Euclid's fifth postulate: start with the empty picture and then add a line. Add a point not on the line. Add a second line through that point. Request that the second line be parallel to the first line. Add a third line through the same point, but different from the second line. The situation is still consistent and many corresponding pictures can be drawn. Then request that the third line be parallel to the first line. No picture corresponds to this situation. The last element of the construction introduced an inconsistency, which corresponds to the following theorem:

> For any three different lines, l_1, l_2, and l_3, and a point p, which lies on l_2, and l_3, but not on l_1, if l_2 is parallel to l_1 then l_3 is not parallel to l_1.

The same theorem can also be stated as:

> For any line l and any point p not on l, there is a unique line that passes through p and is parallel to l.

In this simple example we can see that adding new relationships gradually constrains the situation until it becomes inconsistent. If the situation before the last addition was consistent, while after addition it is inconsistent, we get a theorem.

Figure 1 shows an example of a situation, and its description for the well-known Pappus' theorem: if the three collinear points A_1, B_1 and C_1 are connected as shown to the three collinear points A_2, B_2 and C_2, then the resulting intersection points A, B and C are collinear. The conclusion of the Pappus' theorem $\mathrm{on}(A, \mathit{line}(B, C))$ follows from that description.

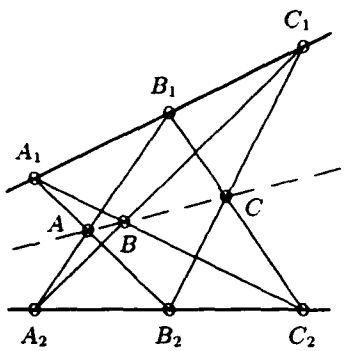
Function symbols, such as *line*, are used to denote objects defined by primitive objects. For example, the term $\mathit{line}(B_1, C_1)$ denotes the line joining the points B_1 and C_1. This works whenever a defined object can be proved to exist uniquely. Moreover, inclusion of circles, for instance, would involve additional relationships, such as $\mathtt{center}(P, C)$ or $\mathtt{tangent}(L, C)$ for describing, respectively, that point P is the center of circle C or that line L and circle C are tangential to each other.

In this paper and in our discovery system, we use the following vocabulary of function and predicate symbols:

Function:	
$\mathit{line}(x, y)$	denotes the line passing through points x and y
Predicates:	
$\mathrm{on}(x, l)$	point x is on line l
$\mathtt{parallel}(l, m)$	lines l and m are parallel and distinct
$\mathtt{intersect}(l, m)$	lines l and m intersect and are distinct

For instance, the atomic statement $\mathrm{on}(A_1, \mathit{line}(B_1, C_1))$ states that the point A_1 lies on the line joining B_1 and C_1. Additional predicates can be easily added.

Definition 1. A *situation description* (or just *description*) is a pair $\langle I, R \rangle$, where I is some set of ingredient points and R is a set of atomic statements and/or negations of such statements about points in I that can be made in the vocabulary.



Ingredient Points:
$A, B, C, A_1, B_1, C_1, A_2, B_2, C_2.$
Relationships:
$\mathbf{on}(A_1, line(B_1, C_1))$
$\mathbf{on}(A_2, line(B_2, C_2))$
$\mathbf{not\ on}(A_1, line(B_2, C_2))$
$\mathbf{not\ on}(B_1, line(B_2, C_2))$
$\mathbf{not\ on}(C_1, line(B_2, C_2))$
$\mathbf{not\ on}(A_2, line(B_1, C_1))$
$\mathbf{not\ on}(B_2, line(B_1, C_1))$
$\mathbf{not\ on}(C_2, line(B_1, C_1))$
$\mathbf{on}(A, line(A_1, B_2))$
$\mathbf{on}(A, line(A_2, B_1))$
$\mathbf{on}(B, line(A_1, C_2))$
$\mathbf{on}(B, line(A_2, C_1))$
$\mathbf{on}(C, line(B_1, C_2))$
$\mathbf{on}(C, line(B_2, C_1))$

Fig. 1. A situation and its description

Situation descriptions can be either consistent or inconsistent. Situations correspond to consistent descriptions, while inconsistent descriptions do not describe any situations.

Detection of Theorems

Pappus' theorem holds in the situation described in Figure 1. The description in Figure 1 includes only the hypothesis (the "if" part) of the theorem and does not contain the conclusion that the points A, B and C are collinear. If the description in Figure 1 is augmented by the atomic statement:

$$\mathbf{not\ on}(A, line(B, C))$$

then the resulting description is inconsistent, i.e. it corresponds to no picture, or equivalently, no coordinates could be assigned to the ingredient points. This suggests the following strategy to arrive at theorems. We start with the simplest situation description containing the empty sets of ingredient points and relationships, and keep expanding it by adding new points and/or relationships. Each description is tested for consistency, and new relationships are added until the description becomes inconsistent. Such a transition from a consistent to an inconsistent description results in a theorem. The consistent description forms the hypotheses of the theorem, and the negation of the new relationship forms its conclusion. Formally, if S is a consistent description and A is a statement such that $S \cup \{A\}$ is inconsistent, then we conclude the theorem:

$$S \rightarrow \neg A.$$

We do not expand inconsistent descriptions any further, because we would obtain large numbers of trivial theorems. Situations can be expanded in various directions, so this

discovery method produces a tree of descriptions. Interesting theorems lie at the leaves of that tree.

The description in Figure 1 can be expanded in many ways, for instance by addition of a new point D or addition of the relationship $\texttt{parallel}(\mathit{line}(A_1, B_1), \mathit{line}(A_2, B_2))$. Some expansions do not result in an inconsistent description. However, an expansion obtained by adding $\texttt{not on}(A, \mathit{line}(B, C))$ results in an inconsistent description leading to Pappus' theorem.

3 Description Generator

In this section we discuss properties desirable for an efficient situation description generator.

For any natural n, the number of descriptions containing exactly n distinct points $p_1, ..., p_n$ is finite. However, many descriptions describe the same situation. For example, for three ingredient points p_1, p_2 and p_3, the descriptions

$$\langle \{p_1, p_2, p_3\}, \{\texttt{on}(p_1, \mathit{line}(p_2, p_3))\} \rangle$$

and

$$\langle \{p_1, p_2, p_3\}, \{\texttt{on}(p_2, \mathit{line}(p_1, p_3))\} \rangle$$

represent the same geometric situation.

Definition 2. Descriptions S and T are *isomorphic* if S can be transformed into T by renaming the ingredient points.

Since the number of descriptions that are isomorphic to any description explodes as factorial ($n!$) with the number of ingredient points, and all isomorphic descriptions are equivalent, it is highly desirable to limit the production of isomorphic descriptions. We do not want our system to spend time proving equivalent theorems, nor do we want to present them as distinct theorems. This leads to the following property:

Property 3. Exactly one description in each isomorphism class should be generated.

As a description is essentially a conjunction of atomic statements, the notion of implication between descriptions can be defined as follows:

Definition 4. A description S *implies* description T if the relationships in S imply those in T.

Intuitively, if S implies T, T describes a simpler (more general) setting than S. Such T should be generated before S, so that all theorems about T are discovered before theorems about S. Theorems which hold for T hold also for S, but according to the rules of the game in mathematics, need not be stated about S because they include some unnecessary assumptions. This leads to the following property:

Property 5. Let $\langle S_1, S_2, \ldots \rangle$ be the order in which descriptions are generated by the generator. If S_i implies S_j then we should have $j \leq i$.

In the next section we will present a generation method to approximate Properties 3 and 5.

4 A Discovery Algorithm

The set of all situation descriptions is recursive and a systematic generation of all descriptions can be easily accomplished by starting from simple descriptions and making them 'grow'. Descriptions can be grown in two orthogonal directions: by imposing more relationships among its objects, or by adding more objects. Only the steps in the former direction can introduce inconsistency.

In addition to an easy implementation, incremental generation of descriptions has other advantages. First, it enables picking out the latest relationship added to a description, if it rendered the description inconsistent, as the negation of the conclusion of the theorem. All other relationships are the hypotheses of the theorem. Second, it helps to enforce Property 5 defined in the previous section. Third, it is necessary for discovery of a theorem in the simplest situation, without any extraneous objects and/or relationships. Further, it is easy to avoid proposing the same, already proven conclusion, in more complex settings, for which it would make a theorem, but would include some unnecessary assumptions. Fourth, the growth of an inconsistent description can be terminated right after it has been detected, to avoid trivial theorems. Fifth, it is an approximation of the way a human discoverer would proceed to uncover theorems.

Let Δ denote the set of all situation descriptions. We define a binary relation $\prec$ on Δ as follows:

Definition 6. $\langle I_1, R_1 \rangle \prec \langle I_2, R_2 \rangle$ iff one of the following is true:

1. $I_2 = I_1 \cup \{P\}$ and $R_2 = R_1$, for some point P not already in I_1;
2. $I_2 = I_1$ and $R_2 = R_1 \cup \{A\}$, for some atomic relationship A not already in R_1.

Let $\prec^+$ be the transitive closure of the above relation. Adding more points or atomic relationships to any description S will yield another description T such that $S \prec^+ T$. It is easy to see that $\prec^+$ is a (irreflexive) partial order on Δ, with least element $S_0 = \langle \emptyset, \emptyset \rangle$. The following proposition is straightforward:

Proposition 7. *For any binary relation* $\mathbf{R}$ *on* Δ, *if* $\mathbf{R}^+ = \prec^+$, *then* $\prec \subseteq \mathbf{R}$.

In other words, $\prec$ is the smallest relation with transitive closure $\prec^+$. Thus it partitions Δ into a sequence of disjoint layers $L_0, L_1, L_2, \ldots$ as shown in Figure 2, where any layer L_k consists of all those descriptions that are at distance k from S_0. (The *distance* of any description $S = \langle I, R \rangle$ from S_0 is $|I| + |R|$.) A breadth-first algorithm that generates descriptions from layer L_k only after all descriptions in layers lower than L_k have been generated, will guarantee globally that no smaller description is generated after a bigger one. That is, if $\langle S_1, S_2, \ldots \rangle$ is the sequence of descriptions generated, then $S_i \prec^+ S_j$ implies $i < j$.

Interface with the Prover

The interface between the discoverer and the prover can be described by a simple dataflow network shown in Figure 3. In the overall organization of the discoverer, the

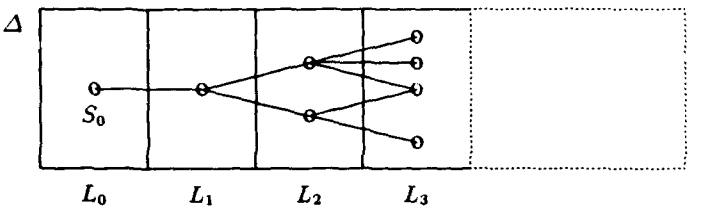


Fig. 2. The layers of situation descriptions

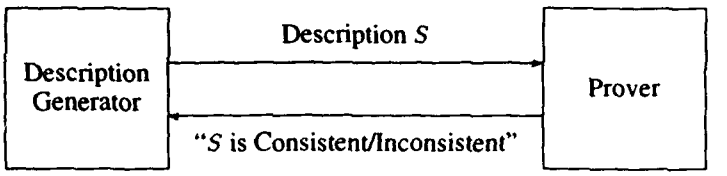


Fig. 3. Interface between description generator and theorem prover

prover is called at those nodes of the situation description graph, when a new relationship has been added and it is not known whether the resultant description is consistent.

The description generator passes on the descriptions to the prover, which recognizes them as either consistent or inconsistent. If the prover recognizes a description S as consistent, the generator creates a theorem.

Let us consider a simple discovery algorithm:

Procedure **DISCOVER**

1 Create a queue Q, consisting solely of the description $S_0 = \langle \emptyset, \emptyset \rangle$.
2 LOOP: Remove the front description from Q. Call this description S.
3 Create all successors of S with respect to the $\prec$ relation.
  Insert each successor into Q, if not already there.
4 If S is already tagged 'inconsistent', assign tag 'inconsistent' to each successor of S in Q.
5 Else use the prover to test consistency of S.
  If S is inconsistent, output the theorem corresponding to S and assign tag 'inconsistent' to each successor of S in Q.
6 Go LOOP.

This algorithm checks "sameness" of descriptions and produces a graph rather than a tree. Observe that the algorithm expands inconsistent descriptions, though no expansion of an inconsistent description is ever submitted to the prover. Such an expansion is useful because descriptions have more than one parent. As long as even one parent of S is inconsistent, S is labeled inconsistent and is not considered by the prover.

Handling Isomorphism

As discussed in the previous section, each description has a large number of isomorphic versions. It is wasteful to invoke the prover for more than one description in any isomorphism class. One way to avoid this is to create all isomorphic descriptions at the time when the first description in a given isomorphism class is generated, and to store all isomorphic equivalents in the hashtable, tagging them at the same time as inconsistent or as consistent. Though this mechanism avoids unnecessary invocations of the prover, creation of all isomorphic equivalents for every description is not very attractive either, and leads to high storage complexity. A better strategy for handling isomorphism is based upon a canonical order in which relationships are added to the situation descriptions, so that only one situation description in each isomorphism class is ever created, but we have not yet implemented this solution.

Handling the Implication Requirement

While the generation strategy outlined above has been easy to implement, the **DISCOVER** algorithm only approximates Property 5, defined in the previous section, which requires that if S_i is implied by S_j (S_j is more specific than S_i) then S_i is generated before S_j. A violation is illustrated by the following example:

$$S = \langle \{A, B, C, D\}, \{\texttt{parallel}(\mathit{line}(A, B), \mathit{line}(C, D))\} \rangle$$
$$T = \langle \{U, V, W, X, Y\}, \{\texttt{not on}(U, \mathit{line}(V, W))\} \rangle$$

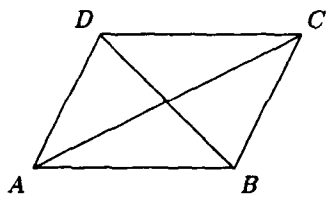
T is implied by S but $T \not\prec S$, because it includes more points. Since S is in the layer L_5 and T is in the layer L_6, the algorithm generates S before T.

5 Example Theorems

In this section we present some simple examples of theorems discovered by an implementation of our method.

Example 1. Let $ABCD$ be a parallelogram. Then the diagonals AC and BD intersect.

This theorem was discovered from the following 4-point description. Since this

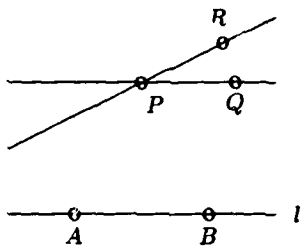


Ingredient Points:
A, B, C, D.
Relationships:
$\texttt{parallel}(\mathit{line}(A, B), \mathit{line}(C, D))$
$\texttt{parallel}(\mathit{line}(A, D), \mathit{line}(B, C))$
$\texttt{parallel}(\mathit{line}(A, C), \mathit{line}(B, D))$

description was tagged 'inconsistent' by the prover, the example theorem has been formed.

Example 2 (Euclid's 5th postulate). Let l be a line and P any point not on l. Then there is a unique line passing through P that is parallel to l

The description corresponding to this theorem is:



Ingredient Points:
$A, B, P, Q, R.$
Relationships:
`not on`$(R, line(P, Q))$
`parallel`$(line(P, Q), line(A, B))$
`parallel`$(line(P, R), line(A, B))$

6 Limitation of the Prover

The underlying prover is based upon Wu's method [10, 11, 3], which takes any set of polynomial equations and inequations as input and determines whether or not they have a complex solution (a solution in complex numbers). Before a description S is presented to the prover it is transformed into a collection C of polynomial equations and inequations. If C is inconsistent, i.e. if it has no complex solution, then it has no solution in real numbers either, and S is inconsistent as well. However, if C is consistent, Wu's method says nothing about the existence of a real solution for C, so that S may have no model in plane geometry. Our system treats S as consistent and when no real solution exists, it misses the corresponding theorem in the process. For this reason, our method is sound but not complete for plane geometry. It is complete only for metric geometry introduced by Wu (see [2]).

7 Conclusions and Future Work

We have presented a mechanism which incrementally generates geometric situation descriptions, tests them for consistency, and generates a theorem for each inconsistent description. Theorems generated by our method do not include redundant conditions in their "if" parts. Our method also avoids theorems which describe isomorphic situations. This way we not only concentrate on interesting and non-redundant theorems, but also we cut down the combinatorial complexity of the task, so that our system can reach more complex theorems within the same computational resources. Though our simple technique does not avoid generation of isomorphic descriptions, a more efficient strategy to deal with this problem is possible. This strategy, based on incremental addition of relationships in a specific canonical order is left for future implementation.

The current theorem discovery method does not use the theorems which have been discovered to alter its situation generation strategy. Our significantly more ambitious goal is to develop a mechanism which uses the theorems to limit the situation generation.

Such a method would begin with no knowledge and learn from its own discoveries. For example, discovery of the simple theorem "on(A,*line*(A, B))" implies that no statement of this type should be used to augment any situation description. Similarly, the discovery "If on(A,*line*(B, C)) then on(B,*line*(A, C))" can be used to avoid generation of many descriptions.

Currently, our system supports the function *line* and three predicates: `on`, `parallel`, and `intersect`, and can discover theorems involving points and lines in these relationships. Expanding the system to handle objects such as circles and relations between lines, points, and circles is easy and will lead to further theorems.

References

1. S. C. Chou. Proving elementary geometry theorems using Wu's algorithm. *Contemporary Mathematics*, 29:243–286, 1984.
2. S. C. Chou. *Mechanical Theorem Proving.* D. Reidel Publishing Company, Dordrecht, Netherlands, 1988.
3. S. C. Chou and X. S. Gao. Ritt-Wu's decomposition algorithm and geometry theorem proving. In *Proceedings of CADE-10*, 1990. Also in LNCS, Vol. 449, pages 207-220, Springer-Verlag, 1990.
4. K. B. Haase. Cyrano-3: an experiment in representational invention. In J. M. Żytkow, editor, *Proceedings of the ML-92 Workshop on Machine Discovery. Aberdeen, U.K.*, pages 153–160, 1992.
5. P. Langley, H. A. Simon, G. L. Bradshaw, and J. M. Żytkow. *Scientific discovery: Computational explorations of the creative processes.* MIT Press, Cambridge, MA, 1987.
6. D. B. Lenat. Automated theory formation in mathematics. *Contemporary Mathematics*, 29:287–314, 1984.
7. Piatetsky-Shapiro and Frawley (Eds.). *Knowledge Discovery in Databases.* AAAI Press, Menlo Park, CA, 1991.
8. Shrager and Langley (Eds.). *Computational Models of Scientific Discovery and Theory Formation.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.
9. M. H. Sims. Empirical and analytic discovery in IL. In *Proceedings of the 4th International Workshop on Machine Learning, Irvine*, 1987.
10. Wen-Tsun Wu. On the decision problem and the mechanization of theorem proving in elementary geometry. *Scientia Sinica*, 21:157–179, 1978.
11. Wen-Tsun Wu. Basic principles of mechanical theorem proving in geometries. *Journal of System Sciences and Mathematical Sciences*, 4(3):207–235, 1984.

Learning Simple Recursive Theories*

A. Giordana, L. Saitta and C. Baroglio

Università di Torino, Dipartimento di Informatica,
Corso Svizzera 185, 10149 TORINO (Italy)

Abstract. The task of learning relations has been concerned, so far, with the acquisition of intensional descriptions of unrelated concepts. However, in many real domains concepts are strictly related to each other and the instances of one of them cannot possibly be recognised without previous recognition of other objects as instances of related concepts. A typical case is the problem of labelling parts of a scene in order to interpret it. This paper extends in several ways the learning relations paradigm; in particular, a new methodology, allowing a recursive theory to be inferred from a set of examples, is presented. The learning algorithm works bottom-up, creating first an acyclic graph that classifies all the instances in the training set. Afterwards, a recursive theory is synthesised from the graph.

1 Introduction

In this paper, an algorithm, called RTL (Recursive Theory Learner), for learning recursive function-free Horn theories, is presented. The first algorithm in the literature for inducing logical programs is due to Shapiro [1], followed later on by others, such as FOIL [2] and CIGOL [3].

However, when several, mutually dependent target relations are to be learned, these approaches suffer from the drawback of an excessive complexity. Let us consider, for instance, the following theory defining a recursive relation for $h_1(x)$:

$$c_1: \quad h_1(y) \wedge \varphi(x,y) \rightarrow h_1(x), \qquad c_2: \quad \xi(x,y) \rightarrow h_1(x) \tag{1.1}$$

Suppose to apply FOIL's learning technique to generate the two clauses c_1 and c_2 from a set of ground instances. Clause c_2 will be learned first, and, then, using c_2 as an operational definition for $h_1(x)$, it will be possible to learn clause c_1. Clause c_1 is recursively applied until no new positive instances of h_1 can be covered. Suppose, moreover, that clauses c_1 and c_2 together do not exhaust the relation h_1 and that a new clause needs now to be learned: c_3: $\psi(x,y) \rightarrow h_1(x)$. Adding c_3 to the set of clauses defining h_1 would require the extensional test for consistency of clause c_1, because now c_1 will also involve $\psi(x,y)$. This leads to a combinatorial explosion of extensional testing.

In the case of the definition of a single predicate $h_1(x)$, the complexity of the above process can still be managed. However, it becomes really unmanageable when mutual recursion occurs. Then, we are proposing another method that does not suffer from this drawback. The method bears some similarities to the ones proposed by Shapiro [1] and by Bierman [4] but introduces some important novelties with respect to the search heuristics which allows the complexity to be dramatically reduced. Algorithm RTL is based on a two phases learning procedure. In the first phase non-recursive definitions

---

* This work has been done in the framework of the ESPRIT Basic Research Action N. 7274 founded by EEC.

are learned by induction. In the second one, the learned theory is generalised by introducing recursive definitions in such a way that non-monotonicity problems do not arise. The algorithm is restricted to learn recursive theories that can be obtained by using only a kind of recursive generalisation, which preserves correctness with respect to the learning set. Nevertheless, this kind of theories is still meaningful for real-world applications. Finally, the method is extended to learn recursive programs stratified with respect to negation [5], which have an acyclic ground graph over the learning set.

2 Basic Notions and Properties

In this section some basic notions and properties will be introduced which are exploited by RTL in order to apply monotonic generalising transformations. Given a non-recursive theory T, we will investigate what kinds of transformations, introducing recursion while preserving the correctness of T with respect to the learning set F, can be applied to T. In the following, x and y denote sets of variables.

Definition 1: Given two relations R, R' of the same arity k, corresponding to the predicates h and h', respectively, R' will be said a *subrelation* of R iff the clause $h' \rightarrow h$ is true. ❑

Definition 1 implies that $R' \subseteq R$.

Definition 2: Given a pair <h, k> of predicates, corresponding to subrelations of R, a clause c: $h \wedge \varphi \rightarrow k$, where h and k have different variables and φ is a conjunctive formula not mentioning any subrelation of R, will be said a *simple clause* over h. ❑

Definition 3: Given a predicate k, corresponding to a subrelation of R, a clause c: $\varphi \rightarrow k$ will be said a *primitive* definition of k, iff φ is a conjunctive formula not mentioning any subrelation of R. ❑

Definition 4: Two clauses c_1 and c_2 are said *correspondent* with respect to a relation R, if they can be made identical by properly renaming the predicates corresponding to subrelations of R, occurring in them. ❑

As an example, the two clauses: $h(y) \wedge \varphi(x,y) \rightarrow k(x)$ and $h'(y) \wedge \varphi(x,y) \rightarrow k(x)$, where h and h' denote subrelations of R, are correspondent with respect to R, because they can be made identical by renaming h as h' or vice-versa. On the contrary, the two clauses: $h(y) \wedge \varphi(x,y) \rightarrow k(x)$ and $h'(y) \wedge \varphi(x,y) \rightarrow k'(x)$ are not correspondent ones.

Definition 5: A theory T_s will be said *simply recurrent* iff all the predicates occurring in the head of the clauses of T_s belong to a predicate set S(R) corresponding to subrelations of R and there exists a bijective mapping from the elements of S(R) to the set of integer $J = \{j \mid 0 \leq j \leq |S(R)|-1\}$ such that:

(a) The predicate $h^{(0)} \in S(R)$ is defined by a set of primitive definitions.

(b) The predicate $h^{(1)} \in S(R)$ is defined by a set $C_J(h^{(1)}, h^{(0)})$ of simple clauses on $h^{(0)}$.

(c) Any other predicate $h^{(i)} \in S(R)$ $\{2 \leq i \leq (|S(R)|-1)\}$ is defined by a set of predicates $C_j(h^{(i)}, h^{(i-1)})$ of simple clauses, whose elements are one-to-one correspondent to the elements of $C_J(h^{(1)}, h^{(0)})$.

(d) No other clause belongs to T_s. ❑

As an example, the following T_s is a simply recurrent theory:

$T_s = \{ \alpha(x,y) \rightarrow h^{(0)}(x);\ h^{(0)}(y) \wedge \varphi(x,y) \rightarrow h^{(1)}(x);$
$h^{(1)}(y) \wedge \varphi(x,y) \rightarrow h^{(2)}(x);\ h^{(2)}(y) \wedge \varphi(x,y) \rightarrow h^{(3)}(x)\}$

Given a set S(R) of subrelations of R, let

$$R_j = \bigcup_{r=0}^{j} EXT(h^{(r)}) \qquad (2.1)$$

be the union of the extensions, evaluated on F, of all the $h^{(r)}$'s $(0 \le r \le j)$. A simply *recurrent theory* shows the interesting property proved in the following:

Theorem 1: Given a simply recurrent theory T_s, if there exists a number n such that the extension on F of the predicate $h^{(n+1)}$ is contained in R_n, then $EXT(h^{(r)}) \subseteq R_n$ $(\forall r \ge n+1)$ In this case T_s will be said *n- complete*.

Proof: By hypothesis, T_s contains the following clauses:

$\alpha(\xi,\psi) \rightarrow h^{(0)}(x)$	$R_0 = EXT(h^{(0)})$
$h^{(0)}(y) \wedge \varphi(x,y) \rightarrow h^{(1)}(x)$	$R_1 = EXT(h^{(0)}) \cup EXT(h^{(1)})$
.....	.....
$h^{(n-1)}(y) \wedge \varphi(x,y) \rightarrow h^{(n)}(x)$	$R_n = EXT(h^{(0)}) \cup ... \cup EXT(h^{(n)})$
$h^{(n)}(y) \wedge \varphi(x,y) \rightarrow h^{(n+1)}(x)$	$EXT(h^{(n+1)}) \subseteq R_n$

The thesis is immediately proved by noticing that for any set of objects $a \in R_n$ there will exist a set of objects b (generated by one of the clauses belonging to T_s) which belongs to R_n either by definition of R_n itself or by hypothesis. Then, being $EXT(h^{(n+1)}) \subseteq R_n$, also the relation $R_{n+k} \subseteq R_n$ $(\forall k \ge 1)$ holds true. *q.d.e.* ■

Theorem 1 guarantees that if the n-th application of a recursive clause does not generate any new instance, then no further application of the same clause will.

Definition 6: Given a simply recurrent theory T_s, let S(R) be the set of subrelations of a same relation R, occurring in T_s. The renaming of all the predicates $h^{(j)} \in S(R)$ with a unique name h will be said a *recursive generalisation* of T_s. The obtained new theory, T, will be called a *simply recursive theory*. ❑

For simply recurrent theories the following theorem holds.

Theorem 2: Given an n-complete, simply recurrent theory T_s, let us rename all the predicates belonging to S(R) as h, obtaining a new theory T. Then, the extension of h on F , in the new theory T, will be equal to R_n.

Proof: By hypothesis, T_s contains the following clauses:

$\alpha(x,y) \rightarrow h^{(0)}(x)$	$R_0 = EXT(h^{(0)})$
$h^{(0)}(y) \wedge \varphi(x,y) \rightarrow h^{(1)}(x)$	$R_1 = EXT(h^{(0)}) \cup EXT(h^{(1)})$
.....	.....
$h^{(n-1)}(y) \wedge \varphi(x,y) \rightarrow h^{(n)}(x)$	$R_n = EXT(h^{(0)}) \cup ... \cup EXT(h^{(n)})$
$h^{(n)}(y) \wedge \varphi(x,y) \rightarrow h^{(n+1)}(x)$	$R_{n+1} = R_n$

By renaming all the $h^{(j)}$'s $(j \ge 0)$ as h, we obtain a new theory T, containing the following clauses: $\alpha(x,y) \rightarrow h(x)$, $h(y) \wedge \varphi(x,y) \rightarrow h(x)$.
The thesis follows from the observation that the original set of clauses in T_s corresponds to the iterative application of the two new recursive clauses in T. Then, $EXT(h) = R_n$. *q.d.e.* ■

Theorem 2 guarantees that recursive generalisation of an n-complete theory does not change its extension.

Theorem 3: Given a theory T, containing an n-complete, simply recurrent subtheory T_s, the recursive generalisation of T_s, obtained by renaming all the predicates in S(R) as h, will not change the extension on F of any other predicate k occurring in T, provided that, for each clause belonging to T-T_s and depending upon some predicate $h^{(i)} \in S(R)$, there exists a correspondent clause for each other predicate $h^{(r)} \in S(R)$ $(0 \le r \le n)$.

Proof: By hypothesis, T_s contains the following clauses:

$\alpha(x,y) \rightarrow h^{(0)}(x)$	$R_0 = EXT(h^{(0)})$
$h^{(0)}(y) \wedge \varphi(x,y) \rightarrow h^{(1)}(x)$	$R_1 = EXT(h^{(0)}) \cup EXT(h^{(1)})$
.....	.....
$h^{(n-1)}(y) \wedge \varphi(x,y) \rightarrow h^{(n)}(x)$	$R_n = EXT(h^{(0)}) \cup ... \cup EXT(h^{(n)})$
$h^{(n)}(y) \wedge \varphi(x,y) \rightarrow h^{(n+1)}(x)$	$R_{n+1} = R_n$

In T, the sub-theory T_s is substituted by: $\alpha(x,y) \rightarrow h(x)$, $h(y) \wedge \varphi(x,y) \rightarrow h(x)$. Let us suppose that the clause $c \equiv h^{(j)}(y) \wedge \psi(x,y) \rightarrow u(x)$ belongs to T-T_s; moreover, no other $h^{(i)}$ $(0 \le i \le n)$ shall occur in the formula ψ.

By hypothesis, a clause correspondent of c exists for each other predicate $h^{(r)} \in S(R)$, i.e.: $h^{(0)}(y) \wedge \psi(x,y) \rightarrow u(x)$, ... , $h^{(n)}(y) \wedge \psi(x,y) \rightarrow u(x)$.

Renaming all the predicates in S(R) implies also renaming the occurrences of any $h^{(j)} \in S(R)$ in the definitions of u. Then, the set of clauses defining u will be substituted by the single clause: $h(y) \wedge \psi(x,y) \rightarrow u(x)$.
By iteratively applying the above clause and remembering that, from Theorem 2, $EXT(h) = R_n$, we can also conclude that the extension of u(x) has not changed.

$$EXT(u) = \bigcup_{r=0}^{n} EXT\left(h^{(r)} \wedge \psi(x,y) \right)$$

q.d.e. ■

Theorem 3 establishes conditions under which recursive generalisation does not require any extensional revision of the theory.

Another problem emerging in learning recursive theories is the occurrence of infinite loops, when the same constants can be bound to variables occurring both in the left-hand and in the right-hand side of a recursive clause. The following theorem states conditions under which loops cannot occur.

Theorem 4: Given an n-complete, simply recurrent theory T_s, the simply recursive theory T, obtained by renaming as h all the predicates belonging to S(R), will have an acyclic ground graph on F iff the extension on F of the predicate $h^{(n+1)} \in S(R)$ is empty.

Proof: For the *if* part of the thesis, we observe that, by hypothesis:
$EXT(h^{(n+1)}) = \varnothing$. We have to prove that T has an acyclic ground graph.

Let us suppose that $\alpha(x,y) \rightarrow h(x)$ and $h(y) \wedge \varphi(x,y) \rightarrow h(x)$ be the recursive definition of h in T. First we observe that, if $EXT(h^{(n+1)}) = \varnothing$, then it is also $EXT(h^{(r)}) = \varnothing$ $(\forall r \ge n+2)$. In fact, the extension $EXT(h^{(r)})$ corresponds to the clause $h^{(r)}$ $(r \ge n+2)$, whose extension is empty, being implied by an already empty premise. Then, the recursion actually stops at the (n+1)-th iteration and there cannot be any infinite loop.

Let us now prove the *only if* part of the thesis. By hypothesis, T_s has an acyclic ground graph on F. This means that no two sets of variables a and b exist, such that: $h(b) \wedge \varphi(a,b) \rightarrow h(a)$ and $h(a) \wedge \varphi(b,a) \rightarrow h(b)$.

In other words, each iteration of the recursive clause $h(y) \wedge \varphi(x,y) \rightarrow h(x)$ will cover, in F, disjoint sets of the relation corresponding to h. As F is a finite set, then $EXT(h) \cap F$ is also a finite set. Then, there will exist an $n \leq |EXT(h) \cap F|$ such that $EXT(h^{(n+1)})$ must be empty.

q.d.e. ■

The inductive algorithm proposed in the next section learns a network of simply recursive theories. We will now investigate under which conditions different simply recursive theories can be merged, without need of extensional tests.

Definition 7: Given two simply recursive theories T_1 and T_2, defining subrelations h_1 and h_2 respectively, T_1 will be said *dependent* on T_2 if it contains at least one clause where h_2, or another subrelation k depending on h_2, occurs in its body. ❑

Definition 8: A set of simply recursive theories will be said a *simply recursive network* SRN iff, for each pair of simply recursive theories (T_1, T_2) belonging to SRN, either T_1 depends on T_2 or T_2 depends on T_1 or both depends on a third one, T_3, belonging to SRN. Moreover, for any $T_i \in STN$, the primitive definitions of h_i should contain at most one occurrence of predicates corresponding to subrelations of another relation R_j for any j. ❑

Definition 9: A simply recursive network SRN is said *complete* iff adding to SRN any correspondent of a clause c, establishing a dependency between a pair of simply recursive theories (T_1, T_2), the extension of SRN on F does not change.

Theorem 5: Given a simply recursive network SRN, if SRN is complete, then renaming to a unique name h all the subrelation of the same relation R will not change the extension of SRN on F.

Proof: The proof is analogous to that of Theorem 2. ■

Finally, we recall the definition of stratified theories due to Apt et al. [6].

Definition 10: A theory T is stratified if there is a mapping *stratum* from the literals L_T, defined in T, to the countable ordinals such that for every clause $L_1 \wedge L_1 \wedge .... \wedge L_n \rightarrow A$ $(n \geq 0)$ in T the following conditions hold for every $1 \leq i \leq n$:
(a) if L_i is positive, say L_i is p, then *stratum*$(A) \geq$ *stratum*(L_i);
(b) if L_i is negative, say L_i is $\neg p$, then *stratum*$(A) >$ *stratum*(L_i). ❑

Informally, a theory is stratified if no recursive loop across negated literals occurs.

3 Learning Recursive Theories

In this section we will introduce a new algorithm, named RTL (Recursive Theory Learner), that learns stratified theories by assembling networks of simply recursive theories.

At first, the algorithm tries to discover recurrent structures in the learning events by iterating clauses discovered by induction. In this way, n-complete simply recurrent theories are discovered and then generalised to simply recursive theories. Afterwards, simply recursive networks (SRN), that are proved complete, are generalised further by applying Theorem 5.

The process can be clarified through an example. Suppose c_1: $h_1^{(0)}(y) \wedge \varphi(x,y) \rightarrow h_1^{(1)}(x)$ and c_2: $h_2^{(3)}(y) \wedge \psi(x,y) \rightarrow h_1^{(0)}(x)$ be two clauses discovered by induction in previous steps. Suppose, moreover, a new definition for predicate $h_2^{(4)}$ is found in some other step; then the new clause $h_2^{(4)}(y) \wedge \psi(x,y) \rightarrow h_1^{(2)}(x)$ will be created, trying to classify new instances of h_1. We will call this operation an *iteration step* on c_2 If the attempt is successful because new instances of h_1 are covered, a new trial is made by iterating clause c_1 and creating clause $h_1^{(2)}(y) \wedge \varphi(x,y) \rightarrow h_1^{(3)}(x)$. Then the process will go on, iterating similar clauses until an iteration step is reached where no new instances of h_1 are covered or an inconsistency is found (i.e. a clause matches facts it should not match). How to deal with inconsistencies is a crucial point and will be discussed later on. In the case that no inconsistency occurs, when the iteration on h_1 ends, a n-complete simply recurrent theory T_s has been discovered, that can be recursively generalised to T.

Now, subrelation h_1^T, defined by T, is evaluated and will be used to start an analogous iterative process for other target relations. The process goes on in this way until no iteration step will cover new instances. We will prove later on that, when this happens, a complete SRN has been found (according to Definition 7). If some target relation is still not completely covered, the induction will be called again in order to find new clauses (primitives and recurrents) to restart the process. However, the goal is to build up SRN as large as possible in order to generalise them into a unique recursive theory according, to Theorem 5. To this aim, clauses where target relation names do not occur negated (positive clauses) are sought until possible by induction. In this way, pre-existing SRNs continue to grow.

When induction is unable to discover positive clauses, the only hope to complete the learning process is to discover clauses where some target relation occurs in negated form (negative clauses). If this is done, a stratum is created in order to prevent the generation of recursive loops across the negated literals. Procedure Stratify, responsible for stratum construction, prevents recursion across strata.

3.1 The Main Algorithm

Algorithm RTL, described in Fig. 1, shows the behaviour described so far. Actually, many details need to be explained. Clause iteration is controlled using specific data structures. In particular, a set (DEPENDENCIES) of clause-schemes is maintained in order to select the clauses to be iterated. A clause-scheme is a clause like c_1 or c_2 where the real names of the target relations H are used in the place of the subrelation names. Moreover, a list ACTIVE(h) is used to schedule the iteration steps for each target relation $h \in H$. When a definition for a subrelation $h^{(j)}$ of h is discovered (by iteration or by induction) for each clause-scheme s depending on h, the procedure Trigger inserts an item $\langle h^{(j)}, s \rangle$ in the ACTIVE list associated to the relation name occurring in the head of s. When a scheme is extracted from the the current ACTIVE list, the procedure Iterate creates a suitable correspondent by renaming the relation names in s.

The inductive activity is scheduled using two lists: TODO and JUSTDONE. The first one contains the target relations which are still to be processed by induction, whereas the second one contains the relations that have been already processed and not yet completely covered. Every time a useful subrelation is learned, the list JUSTDONE is merged into TODO in order to give another chance to induction to exploit the new information just made available.

Algorithm RTL:
Let DEPENDENCIES be the set of mutually dependent clause schemes. Let, moreover, a queue ACTIVE(h_i) of pairs: <subrelation, clause-scheme> be defined for each target relation h_i.
Let TODO and JUSTDONE two sets of target relations used to schedule the inductive activity among the target relations.
Set DEPENDENCIES , ACTIVE(h_i) ($1 \le i \le N$), JUSTDONE = $\emptyset$
Set TODO = H
while TODO $\neq \emptyset$ **do**
  if $\exists h_i$ such that ACTIVE(h_i) $\neq \emptyset$ **then**
    Set ACTIVE = ACTIVE(h_j) being ACTIVE(h_j) chosen according to some preference order established among the non empty ones.
    Set h = h_j
  else Set h = first(TODO) and remove it from TODO
    Set ACTIVE = ACTIVE(h)
    Ph = Newname(); Induce some primitive definition ψ for Ph
    Ph = Newname(); Induce some recurrent φ for Ph
    Update DEPENDENCIES with ψ and φ; Insert <Ph, φ> in ACTIVE
  while ACTIVE $\neq \emptyset$ **do**
    Set φ = First(ACTIVE); Set Ph = Newname(); Iterate(φ, Ph)
    if an inconsistency occur **then** refine(φ)
      if inconsistency remains **then** *Stratify before the last iteration of* φ
    if φ is a recurrent on h and new instances have been covered
      then append <Ph, φ> to ACTIVE
  Set Ph = Newname();*Apply recursive generalization for Ph*
  Trigger DEPENDENCIES using Ph and Update ACTIVE lists
  Set group mark on Ph
  Update the global extension $EXT_g(h)$ for h
  if h is not completely covered **then** Append h to JUSTDONE
  if new instances have been covered in the last cycle **then**
    Append JUSTDONE to TODO
    Set JUSTDONE = $\emptyset$
  else if TODO = $\emptyset$ **then**
    if JUSTDONE $\neq \emptyset$ **then**
      Try to induce a negative definition ψ for some h in JUSTDONE
      Stratify before ψ
      if successful **then**
        Append JUSTDONE to TODO
        Set JUSTDONE = $\emptyset$
    else *stratify*
end

Fig 1. Abstract scheme of Algorithm RTL for inducing recursive theories.

Now the question is: which predicates should induction use in order to discover new clauses? The solution adopted here is based on a heuristics aimed at generating an SRN that can be proved complete without further extensional tests.

First of all, we notice that predicates denoting subrelations obtained in strata below the current one (after generalisation) can be used as legal primitive predicates in addition to the initial set of predicates P. In fact, the corresponding definitions will not be modified any further.

Second, all predicates denoting subrelations defined by simply recursive theories in the current status could be legally used as primitive predicates. However, one of the problems that can arise is the excessive increasing of the predicates that can be used in the induction phase. On the other hand, we are interested in constructing an SRN possibly complete. This means that all possible correspondents to clauses establishing dependency relations between simply recursive theories should always be tried.

A method, semantically equivalent but computationally much more effective to check all the correspondents of a clause c (generated by induction) consists in letting induction use as primitive predicate, not the single subrelations, but a relation h_2^g defined as the union of all subrelations of h_2 already found in the stratum. In this way, each clause of the type $h_2^g(y) \wedge \psi(x,y) \rightarrow h_1^{(j)}(x)$, found by induction, will be equivalent to the whole set of correspondents obtained by substituting $h_2^g(y)$ with the single subrelations $h_2^{(i)}$ already existing in the stratum. Relation h_2^g will grow as long as the process continues; then different names have to be used to distinguish its different instances.

Therefore, we are now able to prove the following theorem about clause iteration process in Algorithm RTL.

Theorem 6: Given Algorithm RTL, when $\forall h \in H$ it is ACTIVE(h) = $\emptyset$ then all STLs in the current stratum are complete.

Proof: Informally the theorem can be proved by observing that, on one hand, each clause generated by induction is implicitly equivalent to the set of all its correspondents generated for all the existing subrelations. On the other hand, each time the definition for a new subrelation $h^{(i)}$ is found, all the possible correspondents for $h^{(i)}$ are automatically iterated by the learning algorithm. Then the process halts because no new facts can be generated by creating new correspondents. ■

3.2 Dealing with Inconsistencies

Possible inconsistencies, detected iterating clauses, are resolved by the procedure Refine. However, the main motivation of the learning strategy proposed here is to escape the complexity inherent in the truth maintenance of the learning process that in presence of recursion can easily become intractable. Eliminating inconsistencies by making a clause more specific, as it is usually done, involves a revision of the extension of all the correspondent clauses instantiated so far. Such a procedure could be very costly. An alternative choice is to forbid recursive generalisation on the group of clauses that proved to have a correspondent inconsistent in some iteration step. Eliminating from a stratum a correspondent clause resulting inconsistent has implicitly this effect, because some SRN will remain incomplete.

Here a compromise between the two alternative has been used. In particular, let c be the clause proved inconsistent; then procedure Refine tries to find a set $\Sigma^{(c)}$ containing one or more new clause-schemes covering the same instances as c in the past iterations but consistent in the current one. If the operation is successful, $\Sigma^{(c)}$ can be substituted to c every where, without requiring an extensional test. Thus, set $\Sigma^{(c)}$ is searched for by running induction under proper constraints.

In the case $\Sigma^{(c)}$ cannot be found, the construction of the current simply recursive theory stops and the creation of a stratum is forced by calling procedure Stratify. In this way, generalisations leading to the inconsistency are prevented.

3.3 Creating Strata

Creating a stratum aims at preventing generalisations, i.e. renaming of subrelations that could merge simply recursive theories external to the stratum with the internal ones. In other words SRNs cannot go across strata boundaries. Strata are created to prevent cycles through negated literals or to prevent generalisations leading to inconsistent definitions.

Let us analyse, first, the problem of avoiding recursion through negated literals. Let us consider, for instance, the following STN containing three simply recursive theories:

$$\begin{aligned} T_1 &= \{\varphi(x,y) \wedge h_1'(y) \rightarrow h_1'(x)\ ;\ \rho(x,y) \rightarrow h_1'(x)\ \} \\ T_2 &= \{\varphi(x,y) \wedge h_1''(y) \rightarrow h_1''(x);\ \xi(x,y) \wedge \neg h_2'(y) \rightarrow h_1''(x)\ \} \\ T_3 &= \{\psi(x,y) \wedge h_2'(y) \rightarrow h_2'(x);\ \sigma(x,y) \wedge h_1'(y) \rightarrow h_2'(x)\ \} \end{aligned} \qquad (3.1)$$

Predicates h_1' and h_1'' cannot be renamed to h_1''' because in the resulting theory:

$$\begin{aligned} T = \{&\varphi(x,y) \wedge h_1'''(y) \rightarrow h_1'''(x);\ \rho(x,y) \rightarrow h_1'''(x)\ ;\ \xi(x,y) \wedge \neg h_2'(y) \rightarrow h_1'''(x); \\ &\psi(x,y) \wedge h_2'(y) \rightarrow h_2'(x);\ \sigma(x,y) \wedge h_1'''(y) \rightarrow h_2'(x)\} \end{aligned}$$

recursive deductions through the negated literal $\neg h_2'(y)$ will occur. However, creating a stratum between T_2 and $\{T_1, T_3\}$ will avoid this kind of problem by keeping h_1' distinct from h_1''.

Let $h_i^{(j)}$ be the subrelation used in negated form by induction; then, procedure Stratify performs the following steps:

a) It individuates the network $SRN(h_i^{(j)})$, $h_i^{(j)}$ belongs to and applies Theorem 5 in order to produce the maximum generalisation as possible without extensional tests.

b) For each different subrelation $h_k^{(r)}$, resulting in step a), the extension on F of $h_k^{(r)}$ is constructed. Then $h_k^{(r)}$ is added to the set of primitive predicates P.

c) The current stratum is updated by removing $SRN(h_i^{(j)})$.

However, things are slightly different when Stratify is called if a clause has shown an inconsistency that procedure Refine could not remove. Suppose the following three simply recursive theories have been generated:

$$\begin{aligned} T_1 &= \{\varphi(x,y) \wedge h_1'(y) \rightarrow h_1'(x);\ \rho(x,y) \rightarrow h_1'(x)\} \\ T_2 &= \{\varphi(x,y) \wedge h_1''(y) \rightarrow h_1''(x);\ \xi(x,y) \wedge h_2'(y) \rightarrow h_1''(x)\} \\ T_3 &= \{\psi(x,y) \wedge h_2'(y) \rightarrow h_2'(x)\ ;\ \sigma(x,y) \wedge h_1'(y) \rightarrow h_2'(x)\} \end{aligned} \qquad (3.2)$$

Suppose, moreover, that the construction of the simply recurrent theory $T_4 = \{\psi(x,y) \wedge h_2^{(1)}(y) \rightarrow h_2^{(2)}(x);\ \sigma(x,y) \wedge h_1'(y) \rightarrow h_2^{(1)}(x)\}$ has been interrupted because clause $\psi(x,y) \wedge h_2^{(1)}(y) \rightarrow h_2^{(2)}(x)$, corresponding to the recursive clause in theory T_3, was found inconsistent. Renaming h_1' and h_1'' to h_1''' would implicitly lead to the introduction of an inconsistency, again because now clauses of theory T_3 will be applied also on all instances of $h_1''(x)$. Let h be the subrelation in the head of the clause that proved to be inconsistent; thus procedure Stratify will operate as in the following:

a') Determine the set S of all simply recursive theories h depends on. (Notice that a non-complete, simply recurrent theory can be seen as a network of simply recursive theories where no recurrent clauses are presents)

a) Individuate a set of SRNs covering all S and for each one apply Theorem 5 in order to obtain the maximum possible generalisation.

b) For each different subrelation $h_k^{(r)}$, resulting in step a), the extension on F of $h_k^{(r)}$ is constructed. Then $h_k^{(r)}$ is added to the set of primitive predicates P.

c) The current stratum is updated by removing $SRN(h_i^{(j)})$.

3.4 Learning Acyclic Theories.

Algorithm RTL learns stratified theories according to Apt definition [5]. However, it may be interesting to obtain theories having an acyclic ground graph on the learning events F [6]. It is easy to prove that, if Theorem 4 is applicable to each simply recurrent theory constructed by Algorithm RTL the ground graph will be acyclic on F. Then it is possible to force the generation of an acyclic theory by restricting recursive generalisation to simply recurrent theories satisfying Theorem 4.

4 Conclusions

In this paper, a new algorithm (called RTL) for learning recursive Horn theories has been described. The adopted approach aims at reducing the complexity due to non-monotonicity of the learning process in presence of mutual dependencies between target relations. In particular, the algorithm learns only recursive dependencies that do not involve revision of clauses already existing. Theories learned by RTL can be stratified according to the definition given by Apt [5] and can be proved acyclic with respect to the learning set. However, this last property can be considered only in a restricted meaning with respect to the general definition given by Apt [6].

References

1. H. Shapiro: Algorithmic Program Debugging. MIT Press: 1982.
2. R. Quinlan: Learning Logical Definitions from Relations. Machine Learning, 5, 239-266 (1990).
3. S. Muggleton and W. Buntine: Machine Invention of First-order Predicates by Inverting Resolution. Proc. Fifth International Conference on Machine Learning. Ann Arbor: 1988, pp. 339-352.
4. A.W. Biermann: The Inference of Regular Lisp Programs from Examples. IEEE Trans. on Systems Man and Cybernetics, 8, 585-600 (1978).
5. K. Apt, H. Blair and A. Walker: Towards a Theory of Declarative Knowledge. Proc. Workshop on Foundations of Deductive Databases and Logic Programming. Washington: 1986.
6. K. Apt and M. Bezem: Acyclic Programs. Proc. 7th. Conference on Logic Programming. Jerusalem: 1990, pp. 617-633.

The Many Faces of Inductive Logic Programming

Luc De Raedt[1] and Nada Lavrač[2]

[1] Department of Computing Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium
[2] Jožef Stefan Institute, Jamova 39, 61111 Ljubljana, Slovenia

Abstract. Inductive logic programming is a research area which has its roots in machine learning and computational logic. A short introduction to this area is given. It investigates the many faces of inductive logic programming and outlines their applications in knowledge discovery and programming. Furthermore, whereas most research in inductive logic programming has focussed on learning single predicates from given datasets using a strong notion of explanation (cf. the well-known systems GOLEM and FOIL), we also investigate a weaker notion of explanation and the learning of multiple predicates. The weaker setting avoids the order dependency problem of the strong notion when learning multiple predicates, extends the representation of the induced hypotheses to full clausal logic, and can be applied to different types of application.

1 Introduction

Inductive logic programming (ILP) [22, 4] can be considered as the intersection of inductive learning and computational logic. From inductive machine learning, ILP inherits its goal: to develop tools and techniques to induce hypotheses from observations (examples) or to synthesize new knowledge from experience. By using computational logic as the representational mechanism for hypotheses and observations, ILP can overcome the two main limitations of classical inductive learning techniques (such as the TDIDT-family [30]):

1. the use of a limited knowledge representation formalism (essentially propositional logic), and
2. the inability to use substantial domain knowledge in the learning process.

The first limitation is important because many problems of various domains of expertise can only be expressed in a first order logic and not in a propositional one, which implies that there are inductive learning tasks for which no propositional learner (and none of the classical empirical learners) can be effective (see e.g. [23]). The difficulty to employ domain knowledge is also crucial because one of the well-established findings of artificial intelligence (and machine learning) is that the use of domain knowledge is essential to achieve intelligent behaviour. First results in applying ILP (cf. e.g. [26, 15, 9]) show that the use of logic as a representation mechanism for inductive systems is not only justified from a theoretical point of view, but also from a practical one.

From computational logic, ILP inherits not only its representational formalism, but also its theoretical orientation as well as some well-established techniques. Indeed, in contrast to most other practical approaches to inductive learning, ILP is also interested in properties of inference rules, in convergence (e.g. soundness and completeness) of algorithms and in the computational complexity of procedures. Furthermore, because of the common representation framework, ILP is relevant to computational logic, deductive databases, knowledge base updating, algorithmic debugging, abduction, constraint logic programming, program synthesis and program analysis, and vice versa.

In this paper, we investigate the different faces of ILP. We first discuss two different semantics for ILP: a strong semantics for ILP as chosen by Muggleton [21] and incorporated in many well-known systems [22, 25, 31, 4, 5, 18], and a weak semantics introduced by Helft [13] and later followed by [12, 7]. It is shown that the strong semantics leads to some problems when learning multiple predicates, and that these problem can be avoided using the weak semantics. The latter also allows for the induction of full first order clauses. The two semantics and the different dimensions are relevant to all ILP techniques. However, the paper is focussed on the so-called refinement (general to specific) techniques used in empirical ILP systems. Other faces of ILP, referring to different dimensions as perceived by users, are also discussed. Finally, we sketch some applications of ILP to knowledge discovery in databases [28] and programming and discuss their relation to the semantics.

The paper is organized as follows: in section 2, we specify the problem of ILP, and study its different faces; in section 3, we survey some ILP techniques for strong ILP; in section 4, we investigate refinement approaches to ILP; in section 5, we study the problem of multiple predicate learning using the strong and the weak settings for ILP; finally, in section 6, we conclude and touch upon related work.

2 Problem specification

Roughly speaking, ILP starts from an initial background theory T and some evidence E (examples). The aim is then to induce a hypothesis H that together with T *explains* some properties of E. In most cases the hypothesis H has to satisfy certain restrictions, which we shall refer to as the bias B. Bias includes prior expectations and assumptions, and can therefore be considered as the logically unjustified part of the background knowledge. Bias is needed to reduce the number of candidate hypotheses. On the other hand, an inappriopriate bias can prevent the learner from finding the intended hypotheses.

In this section, we shall recall some logic programming concepts and use them to formally define the notions of hypothesis, theory, and evidence. Furthermore, we shall discuss two alternative semantics for ILP: the classical one, defined by [21], and the alternative one of [13, 12, 7].

2.1 Some Logic Programming Concepts

Definition 1. A clause is a formula of the form $A_1, ..., A_m \leftarrow B_1, ..., B_n$ where the A_i and B_i are positive literals (atomic formulae).

The above clause can be read as A_1 or ... or A_m if B_1 and ... and B_n. All variables in clauses are universally quantified, although this is not explicitly written. Extending the usual convention for *definite clauses* (where $m = 1$), we call $A_1, ..., A_m$ the *head* of the clause and $B_1, ..., B_n$ the *body* of the clause. A *fact* is a definite clause with an empty body, ($m = 1$, $n = 0$). Throughout the paper, we shall assume that all clauses are *range restricted*, which means that all variables occurring in the head of a clause also occur in its body.

An *example* is a ground fact together with its truthvalue in the intended interpretation; positive examples are true; negative ones are false. The sets of positive and negative examples will be denoted as E^+ and E^-; $E = E^+ \cup E^-$. In ILP, the background theory T and hypotheses H are represented by sets of clauses. For simplicity, we shall mainly focus on definite clauses for representing hypotheses and theories. Nevertheless, parts of our discussion extend to the more general normal program clauses [19], which are sometimes used in ILP systems [18, 31]. Language bias imposes certain syntactic restrictions on the form of clauses allowed in hypotheses; for example, one might consider only constrained clauses [18]; these are clauses for which all variables occurring in the body also occur in the head. Other restrictions frequently employed include abstract languages [4], determinations [33], schemata [14] and ij-determination [25].

Inductive logic programming can now be defined as follows:

Given:

- a set of examples E
- a background theory T
- a language bias B that defines the hypotheses space
- a notion of *explanation* (a *semantics*)

Find : a hypothesis H that satisfies the language bias and explains the examples E with respect to the theory T.

Let us now define the strong and weak settings of ILP, which determine different notions of explanation and semantics. The terms strong and weak are due to Flach [12]. Current research on ILP mainly employs the strong setting [21]:

Definition 2. The strong setting of ILP, restricts H and T to sets of definite clauses, and employs the strong semantics: $T \cup H \models E^+$ and $T \cup H \not\models E^-$[3].

As an illustration, let T = {bird(tweety), bird(oliver)}, E^- = $\emptyset$, and E^+ = {flies(tweety)}. A valid hypothesis in the strong setting is c_1 = flies(X) $\leftarrow$ bird(X).

[3] For convenience, $T \cup H \not\models E^-$ is used as shorthand for $\forall e \in E^- : T \cup H \not\models e$.

Clause c_1 may contribute to a solution because it is ***consistent***, i.e. for any substitution θ for which $head(c_1)\theta$ is false, $body(c_1)\theta$ is also false. Notice that c_1 realizes an inductive leap because T together with c_1 entails flies(oliver).

Definition 3. The weak setting of ILP, restricts T to a set of definite clauses, H to a set of (general) clauses, E to positive examples, and employs the weak semantics: $Comp(T \cup E) \models H$.

$Comp(K)$ denotes Clark's completion of the knowledge base K [3]. The weak semantics of ILP is related to integrity checking in deductive databases (cf. [32]).

To illustrate the weak setting, reconsider the above illustration. In the weak setting, clause c_1 is not a solution because there is a substitution $\theta = \{X \leftarrow oliver\}$ for which $body(c_1)\theta$ is true and $head(c_1)\theta$ is false. However, the clause $c_2 =$ bird(X) $\leftarrow$ flies(X) is a solution because for all X for which flies(X) is true, bird(X) is also true. This shows that the weak setting does not hypothesize properties not holding on the example set. Therefore the weak semantics realizes induction by ***deduction***. Indeed, the induction principle of the weak setting states that the hypothesis H, which is ***deduced*** from the set of ***observed*** examples E and the theory T, holds for ***all*** possible sets of examples E'. This realizes generalization beyond the observed examples. As a consequence, properties derived in the weak setting are more certain than those derived in the strong one.

The differences between the strong and the weak setting are related to the closed world assumption. In most applications of strong ILP [15, 26], only the set of positive examples is specified and the set of negative examples is derived from this by applying the closed world assumption. In our illustration, this results in $E^- = \{flies(oliver)\}$. Given this modified E^-, clause c_1 cannot contribute to a solution in the strong setting because for $\sigma = \{X \leftarrow oliver\}$, $head(c_1)\sigma$ is false while $body(c_1)\sigma$ is true. If on the other hand, we ignore the difference between theory and examples by considering the problem where $T = \emptyset$, $E^+ =$ {flies(tweety), bird(tweety), bird(oliver)}, and $E^- =$ {flies(oliver)} (cwa), clause c_2 is also a solution in the strong setting. Intuitively, this shows that solutions to problems in the strong setting, where the closed world assumption is applied, are also valid in the weak setting. Remember from the database literature that applying the closed world assumption or Clark's completion of the database is only justified when the ***universe*** defined by the examples together with the theory is completely described (cf. also example 1). For example, in a medical application, all patients in the database should be completely specified, which means that all their symptoms and diseases should be fully described. Notice that this is different from requiring that the *complete* universe is described (i.e. all possible patients).

Solutions in the strong setting with the closed world assumption are also solutions in the weak setting. The opposite does not always hold and this reveals the other main difference between the two settings. In the strong setting, the induced hypothesis can always be used to replace the examples because theory and hypothesis entail the observed examples (and possibly other examples as well). In the weak setting, the hypothesis consists of a set of properties holding for

the example set. There are no requirements nor guarantees concerning prediction. For instance in the weak setting, clause c_2 was a solution for T = {bird(tweety), bird(oliver)} and E^+ = {flies(tweety)}. Nevertheless, *it cannot* be used to predict the example in E^+.

The strong and weak faces of ILP are further illustrated by example 1 in a programming context and example 2 in knowledge discovery.

Example 1. (Programming)
Let E = {sort([1],[1]); sort([2,1,3],[1,2,3]); ¬ sort([2,1],[1]); ¬ sort([3,1,2],[2,1,3])}

and let T contain correct definitions for the predicates permutation/2, which is true if the second argument is a permuted list of the first argument, and sorted/1, which is true if its list-argument is sorted in ascending order. Given the strong setting, a possible solution to the inductive logic programming task could consist of the following clause:

sort(X,Y) ← permutation(X,Y),sorted(Y) (c_3)

In the weak setting, using E^+ only, a solution could consist of the following clauses:

sorted(Y) ← sort(X,Y)
permutation(X,Y) ← sort(X,Y)
sorted(X) ← sort(X,X)

Whereas the strong setting results in a program for sorting lists, the weak setting results in a partial specification for the involved predicates. Notice that clause c_3 does not hold in the weak setting because the definitions of permutation and sorted also hold for terms not occurring in E (which means that Clark's completion is not justified, cf. above). On the other hand, if we generalize the notion of a positive example to a definite clause and replace the evidence E by a *correct* definition of the sort predicate (using for instance the definition of quick-sort), clause c_3 holds. This illustrates that the weak setting is applicable to reverse engineering.

Example 2. (Knowledge Discovery)
Suppose we have a database containing weather observations (cf. [36]). Assuming a simplified format, each observation could be described by a fact ob(O,T,H), where O is the label of the observation representing time, T the temperature of the observation, and H the humidity. The general description of the weather observations O could be encoded in predicates such as rain(O), and snow(O). The background theory could then contain definitions for the predicates next(O1,O2) *succeeding* when $O1$ and $O2$ are two subsequent observations; in-temp(O1,O2), in-hum(O1,O2) succeeding when temperature/humidity in $O2$ is increased relative to $O1$.

Let us assume the aim is to learn properties of the predicates rain and snow. In the strong ILP setting, one would start from the set of all positive examples

for these predicates, apply the closed world assumption and use the result as evidence. H could then be:

rain(O) ← ob(O,very-high,very-high)
snow(O) ← next(P,O), ob(P,very-low,high)
snow(O) ← next(P,O), in-temp(O,P), rain(P)

Using the weak setting, one can start learning from the given database (without a need to generate the negative examples) and derive a hypothesis including the above clauses and:

← rain(O),snow(O)
rain(O), snow(O) ← ob(O,T,high)
ob(O,very-low,very-high),ob(O,very-low,high) ← snow(0)

In knowledge discovery, it frequently occurs that the data are noisy, i.e. that they contain random errors. When starting from noisy data, the criteria of both the weak and strong setting should be relaxed and a limited number of mismatches among hypotheses and examples should be tolerated [17, 10].

2.2 Dimensions of ILP

Whereas the above problem specification sketches the faces of inductive logic programming in logical terms, practical ILP systems can be classified in four main dimensions, from a user perspective:

- *Empirical versus incremental.* This dimension describes the way the examples E are obtained. In empirical ILP, the evidence is given at the start and not changed afterwards, in incremental ILP, the user supplies the examples one by one, in a piecewise fashion.
- *Interactive versus non-interactive.* In interactive ILP, the learner is allowed to pose questions to the user about the intended interpretation. Usually these questions query for the intended interpretation of examples or clauses.
- *Predicate invention allowed or not.* Predicate invention denotes the process whereby entirely new predicates (neither present in E nor T) are induced. Predicate invention results in extending the vocabulary of the learner and may therefore facilitate the learning task.
- *Single versus multiple predicate learning.* In single predicate learning, the evidence contains examples for only one predicate and the aim is to induce a definition for this predicate; in multiple predicate learning, the aim is to learn a set of possibly interacting predicate definitions.

In table 1, we sketch some well-known ILP systems along these four dimensions. The ILP systems sketched are: MIS [34], CLINT [4], MOBAL [14], CIGOL [24], FOIL [31], GOLEM [25], and LINUS [18]. From the table it follows that most of the systems are either *incremental multiple* predicate learners or *empirical single*

predicate learners[4]. This means that essentially none of these ILP systems is applicable to knowledge discovery in databases. Knowledge discovery typically requires an empirical setting (all data are given in the database) and involves multiple predicates (as the different predicates in the database should be related to each other). Two novel extensions of FOIL, that can be regarded as empirical multiple predicate learners or knowledge discovery systems, will be discussed in section 5. First, however, we given an overview of some ILP techniques focussing on the strong setting, for which there are more results.

system	Emp/Inc	Int/Nin	Pri/Npr	Sin/Mul
MIS	Inc	Int	Npr	Mul
CLINT	Inc	Int	Pri	Mul
MOBAL	Inc	Nin	Pri	Mul
CIGOL	Inc	Int	Pri	Mul
FOIL	Emp	Nin	Npr	Sin
GOLEM	Emp	Nin	Npr	Sin
LINUS	Emp	Nin	Npr	Sin

Table 1: dimensions of ILP.

3 ILP Techniques in the strong ILP setting

As is the case for most problems of artificial intelligence ILP can be regarded as a search problem. Indeed, the space of possible solutions is determined by the syntactical bias. Furthermore, there is a decision criterion ($T \cup H \models E^+$ and $T \cup H \not\models E^-$) to check whether a candidate is a solution to a particular problem. Searching the whole space is clearly inefficient, therefore structuring the search space is necessary. Nearly all symbolic inductive learning techniques structure the search by means of the dual notions of generalization and specialization [20, 6]. For ILP, there are syntactical [29] and semantical (logical) definitions [27, 2] of these notions:

Definition 4. (Semantic Generalization) A hypothesis H_1 is semantically more general than a hypothesis H_2 with respect to theory T if and only if $T \cup H_1 \models H_2$.

Definition 5. (Syntactic Generalization or θ-subsumption) A clause c_1 (a set of literals) is syntactically more general than a clause c_2 if and only if there exists a substitution θ such that $c_1\theta \subseteq c_2$.

Plotkin has proved that the syntactic notion induces a lattice (up to variable renamings) on the set of all clauses. Notice that when a clause c_1 is syntactically more general than a clause c_2 it is also semantically more general. Clause false is maximally general for both notions of generalizations.

Both *is more general than* relations are useful for induction because:

[4] FOIL and GOLEM should not be regarded as multiple predicate learners, cf. [8] and section 5.1.

- when generalizing a hypothesis H_1 to H_2, all formulae f entailed by the hypothesis H_1 and theory T will also be implied by hypothesis H_2 and theory T, i.e. $(T \cup H_1 \models f) \rightarrow (T \cup H_2 \models f)$.
- when specializing a hypothesis H_1 to H_2, all formulae f logically not entailed by hypothesis H_1 and theory T will not be implied by hypothesis H_2 and theory T either, i.e. $(T \cup H_1 \not\models f) \rightarrow (T \cup H_2 \not\models f)$.

The two properties can be used to prune large parts of the search space. The second property is used in conjunction with positive examples. If a clause does not imply a positive example, all specializations of that clause can be pruned, as they cannot imply the example. The first property is used with negative examples.

Most inductive learners use one of the search strategies below (cf. [6]):

- *General to specific* learners start from the most general clauses and repeatedly specialize them until they no longer imply negative examples; during the search they ensure that the clauses considered imply at least one positive example. Refining clauses is realized by employing a refinement operator, which is an operator that computes a set of specializations of a clause.
- *Specific to general* learners start from the most specific clause that implies a given example; they will then generalize the clause until it cannot further be generalized without implying negative examples. Very often, generalization operators start from a clause and a positive example not implied by the clause; they then compute the starting clause (the most specific clause implying the example) for the example and compute the least general generalization of the two clauses (cf. [29]).

Both techniques repeat their procedure on a reduced example set if the found clause by itself does not imply all positive examples. They use thus an iterative process to compute disjunctive hypotheses consisting of more than one clause. Hypotheses generated by the first approach are usually more general than those generated by the second approach. Therefore the first approach is less cautious and makes larger inductive leaps than the second. General to specific search is very well suited for empirical learning in the presence of noise because it can easily be guided by heuristics. Specific to general search strategies seem better suited for situations where fewer examples are available and for interactive and incremental processing. In the rest of this paper, because of space restrictions, we will only discuss general to specific learning; for specific to general learning, we refer to [22, 4]. Furthermore, because there are more results for strong ILP than for weak ILP, we start with the former.

4 Refinement approaches to strong ILP

Central in general to specific approaches to ILP, is the notion of a refinement operator (first introduced by [34]):

Definition 6. A refinement operator ρ for a language bias B is a mapping from B to 2^B such that $\forall c \in B : \rho(c)$ is a set of specializations of c under θ-subsumption[5].

We call the clauses $c' \in \rho(c)$ the refinements of c. In general, refinement operators depend on the language bias they are defined for. As an example, consider the refinement operator for clausal logic of definition 7:

Definition 7. Clause $c' \in \rho(c)$ iff (1) $c' = head(c), l \leftarrow body(c)$, or (2) $c' = head(c) \leftarrow body(c), l$, or (3) $c' = c\theta$; where θ is a substitution and l a positive literal. Refinements of type (1) are called ***head refinements*** of c; refinements of type (2) ***body refinements***.

Using refinement operators, it is easy to define a simple general to specific search algorithm for finding hypotheses. This is done in algorithm 1, which is basically a simplification of the FOIL algorithm of [31]. FOIL learns DATALOG clauses using a refinement operator for DATALOG. In FOIL's repeat loop different clauses are induced until all positive examples for the considered predicate are implied by the hypothesis. Once a clause is added to the hypothesis, all positive examples implied by that clause are deleted from the set of positive examples. To find one clause, FOIL repeatedly applies a refinement operator (a variant of that in definition 7) until the clause does not imply any negative example for the predicate. FOIL is heuristically guided as it does not consider all refinements of a too general clause, but only the *best* one (the criterion is based on information theory, see [31]). This amounts to a ***hill-climbing*** search strategy. A variant of this algorithm employing beam-search is considered in [10].

FOIL is together with GOLEM [25], which works specific to general instead of general to specific, one of the best known empirical ILP systems. Both systems are very efficient and have been applied to a number of real-life applications of *single*-predicate learning [10, 36, 1, 26, 15, 11]. However, because they employ the strong ILP setting, they suffer from some problems when learning multiple predicates (see [8] and below).

5 Learning multiple predicates

5.1 Strong ILP setting

Examining the problem specification of the strong ILP setting reveals that the clauses in a hypothesis are not independent of each other. Indeed, consider clauses c_1 and c_2. Assume that $T \cup \{c_i\}$ $(i = 1..2)$ imply some positive and no negative examples. So, both c_i could contribute to a solution in the strong setting. Therefore one might consider a hypothesis H containing $\{c_1, c_2\}$. Although neither c_1 nor c_2 implies negative examples, it may be that their conjunction $T \cup \{c_1, c_2\}$ does. This reveals that in the strong ILP setting different clauses of

[5] Sometimes $\rho(c)$ is defined as the set of ***maximal*** specializations of c under θ-subsumption, cf. [34, 7]

```
for all predicates p occurring in E do
        P := the set of all positive examples for p
        N := the set of all negative examples for p
        hypothesis H := ∅
        repeat
                clause c := p(X1, ...Xn) ←; where all Xi are different variables
                while T ∪ {c} implies negative examples from N do
                        build the set S of all refinements of c
                        c := the best element of S
                endwhile
                add c to H
                delete all examples from P implied by T ∪ {c}
        until P is empty
endfor
```

Algorithm 1: A simplified FOIL.

a hypothesis may interact. Because of these interactions, learning multiple predicates in the strong ILP setting is necessarily order dependent as the meaning of one clause depends on the other clauses in the hypothesis. Therefore the order of learning different clauses affects the results of the learning task and even the existence of solutions. As an extreme case, a multiple predicate learning task may only be solvable when clauses (or predicates) are learned in a specific order (because of bias effects and interactions between clauses). In less extreme situations, certain orders may result in more complex (and imprecise) definitions whereas better orders may yield good results.

Present ILP learners in the strong setting provide little help in determining the next clause or predicate to be learned. Indeed, the incremental systems (e.g. MOBAL [14]) rely on the user as they treat the examples in the given order, the interactive ones (CIGOL [24], CLINT [4], and MIS [34]) also generate queries to further simplify the problem, whereas the empirical ones (such as GOLEM [25] and FOIL [31]) leave the problem to the user. Consider for instance the FOIL algorithm outlined in algorithm 1. This algorithm checks only that individual clauses are consistent, not that the hypothesis as a whole is consistent with the negative examples. These and other problems with existing strong ILP learners and multiple predicate learning are discussed in detail in [8].

An attempt to alleviate the above problems is incorporated in the MPL system of [8]. The MPL algorithm is an adaptation of FOIL using the following principles:

- MPL performs a hill-climbing strategy on hypotheses: in each step of this cycle it induces a clause; if adding the induced clause to the current hypothesis results in inconsistencies, MPL will remove some clauses from the current hypothesis; otherwise it will continue with the updated current hypothesis to learn clauses for unimplied positive examples, if any.

- To learn one clause, MPL employs a beam-search similar to mFOIL [10]. The only modification is that rather than storing (definite) clauses and computing their best refinements, MPL keeps track of bodies of clauses, and selects at each step the best body refinements. The quality of a body is defined here as the quality of the best clause in its head refinements. Computing the body first and filling in its head later allows to dynamically determine the next predicate to be learned.
- When estimating the quality of a clause, MPL does not only take into account the examples of the predicate in the head of the clause, but also the other ones. This reduces the number of overgeneralizations.

Preliminary experiments have shown that MPL correctly induces hypotheses for some multiple predicate learning problems beyond the scope of the current empirical ILP approaches, cf. [8].

5.2 Weak ILP setting

Many of the above sketched problems disappear when adopting the weak ILP setting. Indeed, the main problems in the strong setting were concerned with order dependency and dependencies among different clauses. Using the weak semantics these problems disappear because $Comp(T \cup E) \models H_1$ and $Comp(T \cup E) \models H_2$ imples $Comp(T \cup E) \models H_1 \cup H_2$. Therefore clauses can be investigated completely independent of each other (they could even be searched in parallel): it does not matter whether H_1 is found first or H_2. Furthermore, interactions between different clauses are no longer important: if both clauses are individual solutions, their conjunction is also a solution. This property holds because the hypothesis is produced deductively in the weak setting. In contrast, the strong ILP setting allows to make inductive leaps, whereby the hypothesis together with the theory may entail facts not in the evidence. As different clauses in the strong setting entail different facts, the order of inducing and the way of combining different clauses in the strong setting affects the set of entailed facts.

A key observation underlying the CLAUDIEN system [7] is that clauses c not entailed by the completed database $Comp(T \cup E)$ are overly general, i.e. there are substitutions θ for which $body(c)\theta$ is true and $head(c)\theta$ is false in the completed database. In section 4, we saw how overly general clauses could be specialized by applying refinement operators. The same applies here. In particular, body refinements will decrease the number of substitutions for which $body(c)$ holds whereas head refinements will increase the number of substitutions for which $head(c)$ holds.

Based on this we designed the CLAUDIEN algorithm (see algorithm 2) to induce clausal theories from databases. CLAUDIEN starts with a set of clauses Q, initially only containing the most general clause false and repeatedly refines overly general clauses in Q until they satisfy the database. CLAUDIEN prunes away clauses already entailed by the found hypothesis at point (1) because they (nor any of their refinements) can result in new information[6]. Furthermore, at

[6] This is verified using Stickel's fast theorem prover [35].

$Q := \{false\}$; $H := \emptyset$;
while $Q \neq \emptyset$ **do**
    delete c from Q
    if $Comp(T \cup E) \models c$
    then **if** $H \not\models c$ (1)
        then add c to H
        endif
    else **for** all relevant $c' \in \rho(c)$ (2) **do**
        add c' to Q
        endfor
    endif
endwhile

Algorithm 2: A simplified CLAUDIEN.

point (2) it tests whether refinements are relevant. Roughly speaking, relevance means that at least one substitution is handled differently by the refinement clause than by its parent. Under certain conditions on the language bias and refinement operator discussed in [7], irrelevant clauses can safely be pruned away. Intuitively, an irrelevant refinement c' of c is a refinement for which c and c' are logically equivalent in $Comp(T \cup E)$. Notice also that CLAUDIEN does not employ a heuristic search strategy, but a complete search of the relevant parts of the search space. Complete search[7] allows to find all properties expressible in the given language bias, thereby the most general hypothesis explaining the database is found.

Example 3. (A CLAUDIEN example)
We ran CLAUDIEN on a database containing family relations father, mother, parent, male, female and human. The bias restricted clauses to contain at most two variables and at most 3 literals. Any combination of predicates in clauses was allowed. CLAUDIEN derived the following theory:

← female(X), male(X)
human(X) ← male(X) (1)
human(X) ← female(X) (2)
female(X),male(X) ← human(X) (3)
← parent(X,X)
parent(X,Y) ← mother(X,Y)
parent(X,Y) ← father(X,Y)
← parent(X,Y),parent(Y,X)
← father(X,Y),mother(X,Y)
father(X,Y),mother(X,Y) ← parent(X,Y)
human(Y) ← parent(X,Y)

[7] In the implementation, we employ depth-first iterative deepening [16].

human(X) ← parent(X,Y)
female(X) ← mother(X,Y)
male(X) ← father(X,Y)

The induced theory contains most of the relevant information about the involved predicates. It could be used as an integrity theory for a database. It also contributes to understanding the domain of family relations as it contains some relevant regularities. Notice also that the theory can be used for predictions. Indeed, if parent(luc,soetkin) and male(luc) is asserted we can derive ¬ mother(luc,soetkin), father(luc,soetkin), female(soetkin) ∨ male(soetkin), human(soetkin), etc. On the other hand, if only father(luc,soetkin) is asserted we can derive human(luc), human(soetkin), female(soetkin) ∨ male(soetkin), male(luc), parent(luc,soetkin), ¬ female(luc). It is easy to see that using the above theory, prediction can start from basically any fact(s) for any set of predicates. This contrasts significantly from the strong ILP setting, where the inferences only go in one direction. Given the usual approaches, the learner would induce only clauses for specific predicates. E.g. the clauses (1) and (2) could be inferred, or alternatively the two normal clauses corresponding to clause (3), i.e. male(X) ← human(X), ¬ female(X) (4) and female(X) ← human(X), ¬ male(X) (5). The strong ILP systems would not allow to induce both (1-2) and (4-5), because the resulting program loops. Therefore the theory induced in strong ILP can only be used in one direction. Given (4-5) one can deduce facts for male and female. Given (1-2), facts for human one can deduce facts for male or female.

6 Conclusions

We have defined and investigated different faces of ILP, focussing on the semantics of ILP. We believe the weak semantics puts several issues in ILP in a new perspective. First, the weak setting allows to induce full clausal theories. Second, the weak setting makes the learning of different clauses order independent, which is especially useful when learning multiple predicates. Third, the weak setting derives properties of examples instead of rules generating examples. Although such properties cannot always be used for predicting the truthvalues of facts, we have seen that the use of full clausal theories can result in predictions not possible using the less expressive strong ILP setting. Fourth, it was argued that the common practice of applying the closed world assumption on the example set in strong ILP, corresponds to the use of Clark's completion and a deductive setting for weak ILP.

On the other hand, when we are interested in predictions, then the strong setting is more appropriate as induced hypotheses in the strong setting can always be used for prediction. Furthermore, although learning multiple predicates in the strong is less efficient, we have presented a preliminary approach to multiple predicate learning in the strong setting.

Acknowledgements This work is part of the ESPRIT Basic Research project no. 6020 on Inductive Logic Programming. Luc De Raedt is supported by the Belgian National Fund for Scientific Research; Nada Lavrač is funded by the Slovenian Ministry of Science and Technology. The authors are grateful to Ivan Bratko, Danny De Schreye, Saso Dzeroski, Bern Martens, Stephen Muggleton, Gunther Sablon for discussions, suggestions and encouragements concerning this work. Special thanks to Maurice Bruynooghe and Peter Flach for suggesting many improvements to an earlier version of this paper.

References

1. I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 385–388. Morgan Kaufmann, 1991.
2. Wray Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:375–399, 1988.
3. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
4. L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.
5. L. De Raedt and M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53:291–307, 1992.
6. L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.
7. L. De Raedt and M. Bruynooghe. A theory of clausal discovery. Technical Report KUL-CW-164, Department of Computer Science, Katholieke Universiteit Leuven, 1993. to appear in Proceedings of the 3rd International Workshop on Inductive Logic Programming.
8. L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. Technical Report KUL-CW-165, Department of Computer Science, Katholieke Universiteit Leuven, 1993. to appear in Proceedings of the 3rd International Workshop on Inductive Logic Programming.
9. B. Dolsak and S. Muggleton. The application of inductive logic programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, 1992.
10. S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, 1992.
11. C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 403–406. Morgan Kaufmann, 1991.
12. P. Flach. A framework for inductive logic programming. In S. Muggleton, editor, *Inductive logic programming*. Academic Press, 1992.
13. N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156. Morgan Kaufmann, 1989.
14. J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

15. R.D. King, S. Muggleton, R.A. Lewis, and M.J.E. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 1992.
16. R. Korf. Depth-first iterative deepening : an optimal admissable search. *Artificial Intelligence*, 1985.
17. N. Lavrač and S. Džeroski. Inductive learning of relations from noisy examples. In Muggleton S., editor, *Inductive Logic Programming Workshop*, pages 495–514. Academic Press, 1992.
18. N. Lavrač, S. Džeroski, and M. Grobelnik. Learning non-recursive definitions of relations with LINUS. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.
19. J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.
20. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
21. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–317, 1991.
22. S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.
23. S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113-118. Morgan Kaufmann, 1989.
24. S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.
25. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*. Ohmsma, Tokyo, Japan, 1990.
26. S. Muggleton, R.D. King, and M.J.E. Sternberg. Protein secondary structure prediction using logic. *Protein Engineering*, 7:647–657, 1992.
27. T. Niblett. A study of generalisation in logic programs. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session On Learning*, pages 131–138. Pitman, 1988.
28. G. Piatetsky-Shapiro and W. Frawley, editors. *Knowledge discovery in databases*. The MIT press, 1991.
29. G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.
30. J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
31. J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.
32. R. Reiter. On asking what a database knows. In J.W. Lloyd, editor, *Computational Logic*, pages 96–113. Springer-Verlag, 1990.
33. S.J. Russell. *The use of knowledge in analogy and induction*. Pitman, 1989.
34. E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT press, 1983.
35. M.E. Stickel. A prolog technology theorem prover: implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, 1988.
36. C. Vermeulen. Een toepassing van automatisch leren op weersvoorspellingen. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1992. in Dutch.

CONSENSUS: A Method for the Development of Distributed Intelligent Systems

Michael Bateman

British Aerospace Defence Ltd,
Warton Aerodrome, Preston PR4 1AX, UK

Sean Martin

Cambridge Consultants Ltd,
Cambridge Science Park, Cambridge CB4 4DW, UK

Andrew Slade

School Engineering and Computer Science, University of Durham,
South Road, Durham DH1 3LE, UK

Abstract. This paper reports on the work of CONSENSUS, a collaborative project funded by the United Kingdom Department of Trade and Industry Advanced Technology Programme. The project has produced a proven engineering method for developing large scale real-time systems with distributed intelligent components. The paper presents an overview of the method that has been developed and illustrates how it has been used to construct a large scale application in the area of air traffic control. The method has also been used to develop a dynamic tactical planning application that reacts to events as they arise, which is also briefly described in the paper.

1 Introduction

Many applications, where new technologies are being deployed, are characterised by:

- increasing quantity of data from sensors and other sources,
- increasing need for interpretation of data,
- reducing response times,
- increasing workload,
- increasing need for planning in response to events as they arise,
- increasing demand for multiplexing, display management and control.

The commercial justification for new systems is usually based on delivering greater throughput or performance without a corresponding increase in staffing levels. This requires the division of responsibility between man and machine to be repartitioned: hence the machine will perform some tasks that may have previously been performed by the man, and thus require intelligent behaviour. There is thus a need for increasing intelligence in applications where new technologies are deployed.

In addition, since most practical applications are inherently distributed or naturally concurrent, there is a need for distributed intelligent behaviour. In most cases a centralised approach is inappropriate thus emphasising the need for distributed intelligence, which, for example, is more resilient in the event of partial system failure.

There has as yet not been an accepted software engineering method for developing real-time distributed knowledge based systems. As Bond and Gasser [3] note:

> "An engineering science perspective on Distributed AI would investigate how to build automated, coordinated problem solvers for specific applications."

The CONSENSUS project addresses this need, and has produced a proven engineering method for developing operational systems with distributed knowledge based components that address the requirements listed above. This provides a sound basis for developing operational systems maximising the benefits delivered by advanced software technology.

2 The Scope of the CONSENSUS Method

The CONSENSUS project involves some fifteen man years of effort shared by three partners over a three year period. The partners are British Aerospace Defence Limited, Cambridge Consultants Limited and the University of Durham, all of whom have extensive experience in concurrent real-time knowledge based engineering applications.

The scope of the CONSENSUS method is illustrated in Figure 1. It concerns the development of large real time systems comprising knowledge based components that need to be engineered into a system that meets its needs. By implication, the scope of the project covers the complete life cycle from concept through to operation, and includes:

- an analysis of the need and functioning of the system,
- derivation of an appropriate architectural design,
- detailed design and implementation,
- verification, validation and testing,
- maintenance and support.

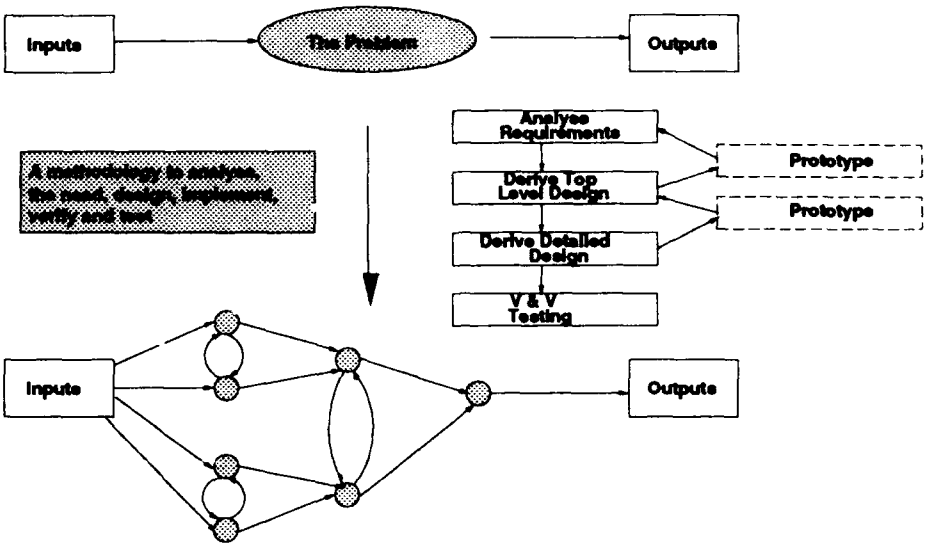


Fig. 1. The Scope of the CONSENSUS Method

The method helps the designer to visualise how distributed intelligence will function in terms of the interaction between knowledge based components and to assess the associated demands for communication.

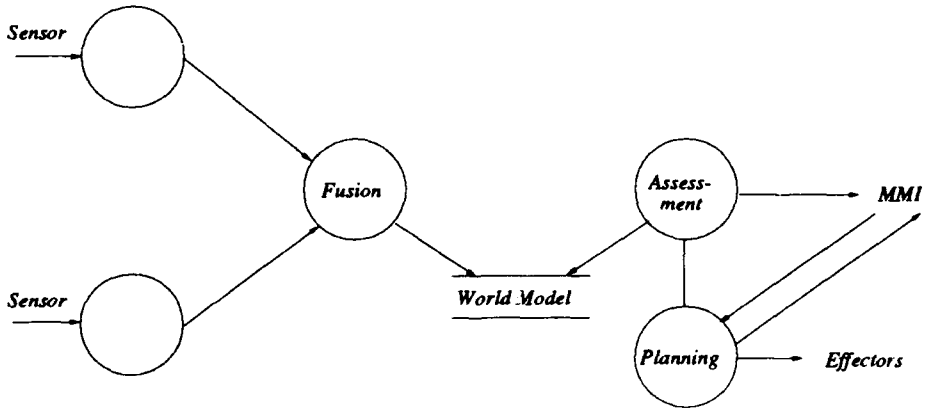


Fig. 2. High Level System Functions

The CONSENSUS approach is suited to applications that exhibit any of the following characteristics, as illustrated in Figure 2:

- data from a variety of sources is used and combined (i.e. data fusion),
- a model of the external world is constructed from this information,

- real time reasoning is required to react to events as they arise,
- off-line reasoning is required for longer term planning,
- there may be several reasoning modules,
- there is a controller for supervising actions,
- there are effectors (i.e. something is controlled or displayed),
- subsystems can be identified that co-operate to fulfil the system goals,
- there may be varying granularity between the subsystems,
- the application is inherently distributed or runs on a distributed network.

Many applications have these characteristics and there are often two high level architectural approaches that are both addressed by the CONSENSUS method:

- a distributed system with centralised functions: where data may be gathered and processed at source but overall coordination is exercised from a control room,
- a distributed system with distributed functions: where, as in telecommunications network management, control is distributed.

3 The CONSENSUS Method

3.1 Overview

The CONSENSUS method is a system specification approach which is made up of a requirements model based on a current software engineering approach for the development of complex software systems and an architecture model based on a distributed blackboard approach. The key points are:

- the requirements model and the architecture model are developed *together*;
- the requirements model defines what the system is to do and is independent of the implementation technology;
- the architecture model defines how the system is to be structured and must take account of how the system is to be implemented.

The requirements and architecture models are developed in parallel, starting with a high level description of the system, and proceeding by refinement and iteration until a detailed and complete definition of the system is produced. This view of systems development is illustrated in figure 3.

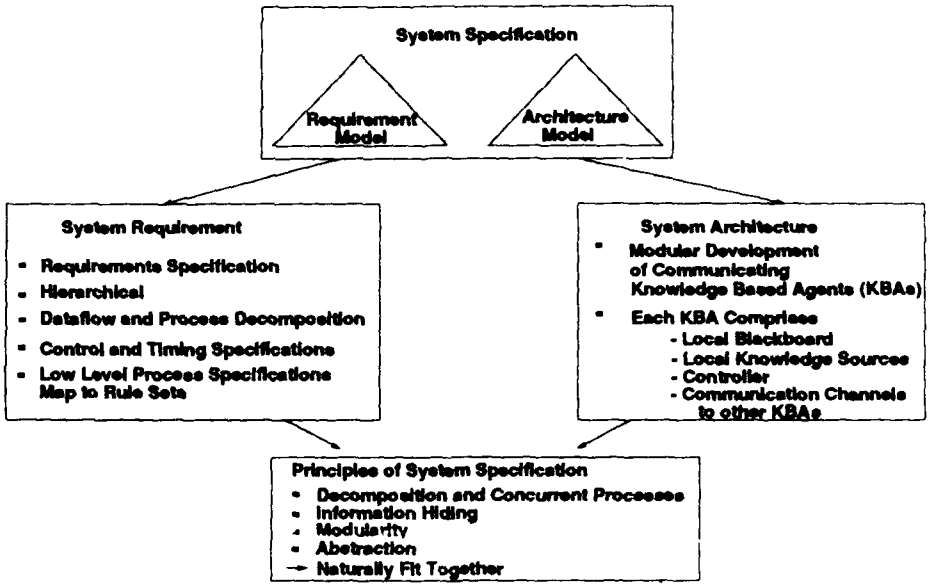


Fig. 3. Overview of the Method

The primary benefit of this approach is that early partitioning and allocation of functions in the system helps identify critical functions which can be prototyped or reappraised.

3.2 The CONSENSUS Requirements model

As illustrated in Figure 3, the approach for developing the requirements model is based on real time structured analysis techniques using hierarchical data flow and process decomposition, together with control and timing specifications, to allow for the requirements of real-time systems.

The behaviour of the system is analysed and decomposed into a set of simpler process models, each of which is characterised by a process definition. In practice these map quite well to rule sets defining explicit knowledge in the implementation.

The process of functional decomposition is taken to the point where the system requirements cannot be refined any further, and a level of detail is reached where implementation (that is to say architectural) considerations come into play.

The process of functional decomposition starts with a top- level view of the system and by a process of iterative refinement produces:

- a hierarchy of data-flow and control-flow diagrams, starting with a system context diagram and followed by as many levels of decomposition as are necesssary, specifying how the system is structured into processes, data-flows and control-flows;
- a set of process specifications defining how inputs are transformed into outputs;
- a set of control specifications defining the interaction of control signals;
- a definition of the objects and their structure;

- a definition of the objects and their structure;
- a set of timing specifications defining the limits in response time allowed between inputs and output.

The above are developed in conjunction with the architecture model, describing and defining the system in terms of the implementation technology. The process of functional decomposition should produce collections of functions, which can be grouped together into separate processes which can work concurrently and collaborate by communication towards the overall goal of the system.

It may also be possible to identify groups of potentially concurrent functions at different degrees of resolution, but the decision on this matter is more dependent on implementation constraints and should be taken from the system architecture model.

3.3 The CONSENSUS Architecture Model

Soon after beginning the requirements model, the developer begins the task of deriving the architecture model, in which constraints imposed by implementation considerations may be more appropriately addressed. The development of the architecture specification is based on a technique for developing distributed blackboard and multi-agent architectures. This allows multiple components of the system to communicate and thereby co-operate towards solving the system's goals.

This approach encourages modular design comprising several independent communicating knowledge based agents (KBAs). Each KBA comprises a local blackboard, accessed by local knowledge sources, a controller and communication channels to other KBAs. The communication between KBAs is constrained to encourage modularity.

The system architecture model is derived in parallel to the requirements model. It is the purpose of the system architecture model to:

- identify the set of agents which make up the system;
- define the information flow between the agents in the system; and
- specify the channels on which the information flows.

This is implemented by using an arrangement similar to that of the requirements model. Again diagrams and supporting textual specifications are used:

- a set of architecture flow diagrams showing the configuration of the knowledge based agents and the data flow between them;
- a set of architecture interconnect diagrams showing the physical interconnection between the knowledge based agents in terms of channels;
- a set of architecture module specifications capturing the allocation of the requirements for each knowledge based agent;
- a set of architecture interconnect specifications detailing the properties and characteristics of each channel between agents.

The grouping of functions which is the result of functional decomposition together with the decision of where to pitch the transition point from functional decomposition to knowledge based agents will yield the division of the overall task into a collection of concurrent cooperating agents. There are a number of factors to consider in the decision of when (in terms of the decomposition) to do the translation from functional decomposition to KBAs:

Although the technological base for the requirement and the architecture models is quite different, the fundamental principles are very similar, and in consequence they do fit together very well to form an integrated approach for developing systems with concurrent knowledge based components. They both share the following principles in common:

- decomposition: into co-operating concurrent processes communicating via defined channels;
- information hiding: whereby individual processes are not concerned with the internal details of other processes;
- modularity: whereby individual processes can be designed, developed and tested in isolation prior to integration with other processes;
- abstraction: allowing high, as well as low, level system specifications.

Although the primary purpose of the CONSENSUS method is for the design of knowledge based components, its sound basis on software engineering principles allows it to be applied to purely procedural implementations and on mixed systems comprising both procedural and knowledge based components.

4 The CONSENSUS Demonstrator Applications

The CONSENSUS method can be illustrated by the development of one of the two demonstrator applications of the project, which is described below.

4.1 Air Traffic Control Demonstrator

The application is based on a report, publicly available at the Civil Aviation Authority library,[2] defining an intelligent system which could act as an assistant to an air traffic controller. The top level of the system is made up of a simulator, an ATC workstation and the interface between the two. The application is implemented in Muse [4]. Muse is a real-time AI toolkit based upon the blackboard model of problem solving, providing a number of knowledge representation styles (rules, demons, objects, and so on). The final implementation uses 27 Muse processes and one C process, communicating with each other through sockets.

The system comprises the components of the ATC workstation and a simulator of controlled aircraft, where the overall organisation is shown in Figure 4:

- the total number of KBAs should not be so great that the communications overheads are prohibitive. KBAs will usually not be simple processes, but complex;
- KBAs should be manageable in size and complexity. If there is too much functionality in a given KBA it should be split up (especially when there is an apparent fracture of functionality);
- if simple processes are closely associated and share data or communicate with each other, then they should be grouped together in a single KBA;
- if simple processes share the same parent then they should probably be in the same KBA;

In the above context it should be borne in mind that simple processes refer to those which have too low a workload to be separate KBAs for reasons of communication overheads. These are not necessarily the same as primitive processes in the requirements model.

In order to derive an architecture model for a CONSENSUS system the specifications in the requirements model need to be translated into specifications for the different KBAs which make up the system, together with a specification for the architecture and communication between them:

- Communication Channels: The data-flows and control-flows of the data- and control-flow diagrams need to be translated into messages passed through predefined communication channels observing the restrictions imposed by the architecture.
- Local Database: All data which is required for processing, the results of processing and messages to and from other KBAs appear in the local database. The data-stores also need to be included either in dedicated KBAs or as a special section within a KBA.
- Knowledge Sources: The actual functionality of the system, as described by the requirements model, will be translated into a collection of knowledge sources which operate on the contents of their local database or by requesting communications with other KBAs.
- Local Controller: The local controllers exert control over the action of the local knowledge sources and act as the postman on behalf of their knowledge sources for communications with other KBAs. If there are any control requirements left after the functional requirements have been translated into knowledge sources, they have to be implemented by the local controller.

3.4 Overall Principles

Superficially, the process of deriving a system specification from a requirements model and an architecture model would appear to be a straightforward one. However, experience with the design for the first demonstrator application (described below) has shown that the transition from functional decomposition used in the requirements model to agent based design used in the architecture model needs guidance. This problem has received considerable attention in the project, and has become a part of the method.

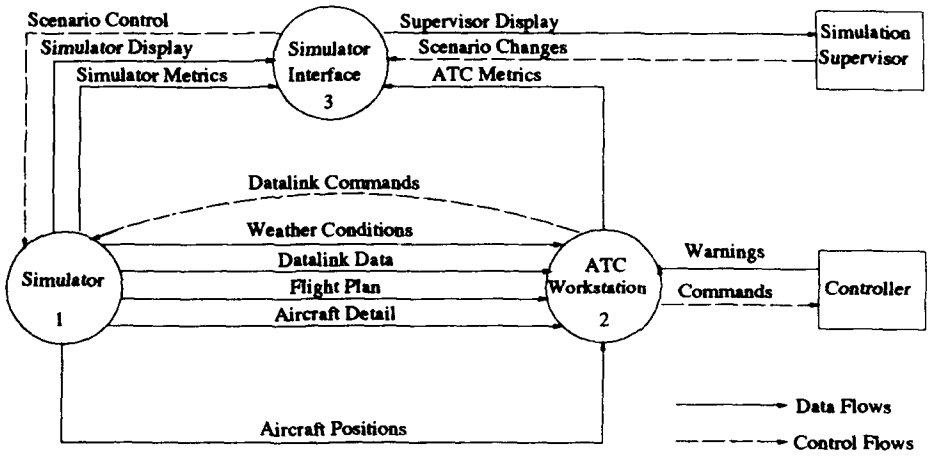


Fig 4. Air Traffic Control Application

The Simulator. The simulator generates scenarios for the ATC workstation and responds to messages from it. There are some pre-defined scenarios and scenario changes can be incorporated from a supervisory console. Performance metrics are also catered for. The simulator (process 1) breaks down into five (top-level) processes. A Traffic Generation process generates the air traffic that appears during a simulation run. A Route Planner generates and updates an aircraft's flight plan. An Aircraft Movement Controller calculates an aircraft's movement from its details, flight plan and the weather conditions. An Advice Checker checks the conflict avoidance advice and information requests coming into the simulator. A Weather Details process manages the weather within the air sector and those sub-sectors that bound the air sector.

The ATC workstation. The ATC workstation (process 2) consists of the following processes. (Compared to the original conception, the functionality has been slimmed down). The Predictor calculates predicted courses of aircraft (up to 20 minutes ahead), and warns of any impending conflicts. It also performs prediction services for the other tools. The Wotifer (from "what if...?") assists the controller in the rapid formulation of a plan for the movement of an aircraft through his sector. For example, it offers default plans to the controller, or fills in plan details. The Aircraft Arrival process sets up the details of an aircraft before it enters the control of the ATC workstation. The Communicator handles the rapid communication of messages for datalink transmission and the display of datalink messages received. It also validates messages to aircraft against the aircraft details. The Monitor monitors aircraft for conformance with flight plan, or for any unusual/abnormal behaviour. It also monitors controller instructions against aircraft capability. The MMI component displays the workings of other components, providing a visual indication of conflicts, deviations from flight plan, recommended routes and so on.

4.2 Design Illustrations.

To show the method at work, the Monitor component (process 2.5) of the ATC workstation will be used as an example. The examples used come from the design documentation and source code of the demonstrator application; (... indicates omitted text).

The Monitor. The Monitor is divided into several sub-processes, as shown in the requirements model diagrams of figures 5.and 6. (Circles represent processes; unbroken lines are data flows; dotted lines are control flows, which basically enable or disable processes; pairs of horizontal lines indicate data stores). The Command Monitor monitors aircraft for conformance with clearances. It needs details of the aircraft's position and the last command issued to it. The Command Checker compares each new command (as it arrives) with the flight plan for the appropriate aircraft. The Flight Plan Override regenerates a last command in cases where an aircraft is off the flight plan and for some reason there is no last command. The Flight Monitor checks aircraft positions against the current flight plan. If the aircraft is not on the flight plan (in four dimensions) within the same degree of accuracy as used by the Command Checker, above, then various validation and warning steps are taken. The Behaviour Monitor checks each aircraft's position and history (previous positions) against the aircraft's performance statistics. If, after taking weather conditions into account, the aircraft appears to be behaving abnormally then various error reports are sent out. The Boundary Monitor keeps track of whether, and when, an aircraft leaves or enters a sector. The New Aircraft Checker determines whether or not an aircraft, which is under the controller's control and is approaching the sector, will enter the sector at the expected position and time.

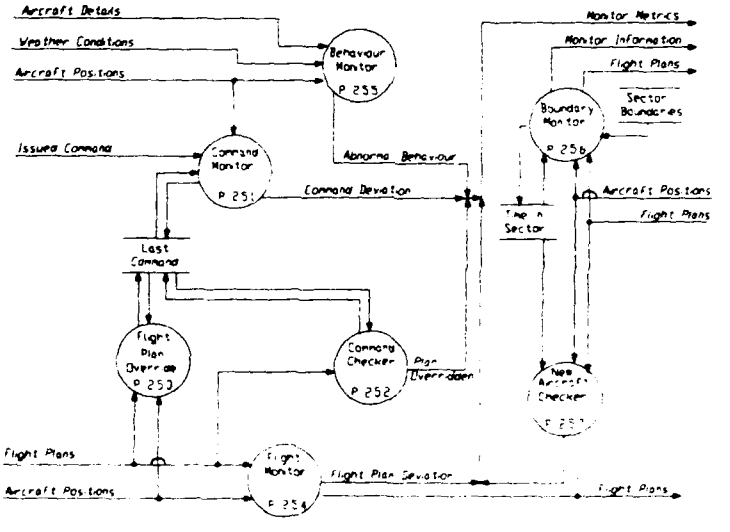


Fig 5. Data Flow Diagram for the Monitor

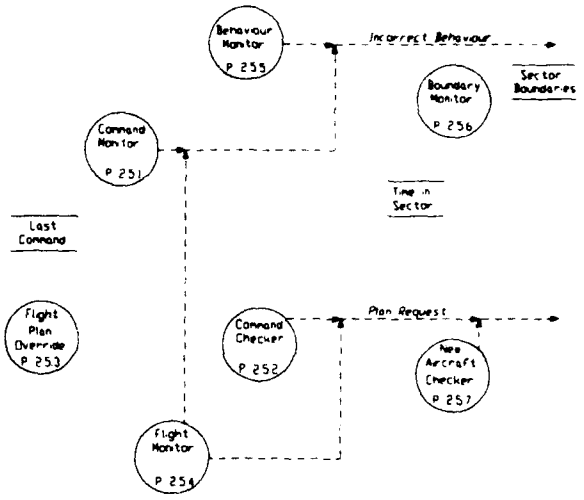


Fig 6. Control Flow Diagram for the Monitor

Requirements: Sample Process Specification. The requirements model specifies what the system is to do. Its main components are data and control flow diagrams, and textual process specifications (PSPECs). An example of a PSPEC, taken from the design document for the Monitor process, is shown below. It states the requirements for the Command Monitor (process 2.5.1). A language standard for the writing of PSPECs was devised as a part of the work of the project. The example borrows from various sources: constant definitions (const); structure accesses (by .) and so on. Braces are used for the introduction of an abbreviation as well as for comments. Some auxiliary function definitions (beginning with Function) are used to extract lengthy pieces of a PSPEC, or pieces found to occur in several PSPECs.

```
Command Monitor 2.5.1: PSPEC

const Epsilon Direction {ED} = 1 (x (degree): allowed
variance in course)    ...
When receive Aircraft Positions {AP}
   if there is a Last Command {LC} [for LC.Id = AP.Id]
then
  ....
  else
    create AP.Id in LC with value := null
  endif
endwhen
```

```
Function Course_Diff(angle1, angle2)
{Given two angles calc. the difference between them,
taking the 0/360 boundary into account}
  angle := abs(angle1 - angle2)
  if (angle > 180) then
    angle := (360 - angle)
  endif
return angle
...
When get Issued Command {IC}
  case IC.Message Type of
    H,S,T,W  :  store Message in LC for AP.Id
        {less Message Id, plus time now}
    P  :    store null in LC for AP.Id
    Otherwise  :  {A,C,F,N,O,U,X} do nothing
  endcase
endwhen
```

Architecture: Design Decisions. The architecture model specifies how the system is to be structured; it is the allocation of available means to fulfil requirements. The CONSENSUS architecture model is built by identifying Knowledge Based Agents within the requirements model. It is recorded by merely grouping process numbers using set notation. For example, to state that sub-processes 2.5.1 and 2.5.5 form the first Knowledge Based Agent (KBA) in the architecture design for the process 2.5, one writes: KBA2_5.1 = {2.5.1, 2.5.5}.

An important part of the design side of the method is the provision of guidelines for KBA identification. A number of factors can influence the architecture design corresponding to a requirements model, because there is some indeterminacy in the notion of an "architecture". In an abstract sense, an architecture states how a requirement will be fulfilled, while in a concrete sense an architecture relates requirements to physical means for meeting them. The notion of a KBA used in the design process is somewhat indeterminate: sometimes a unit in an abstract architecture for distributed problem solving, sometimes a processor, or process running on a processor. There may also be uncertainty about the number of available processors, or practical limit on number of processes per processor. The indeterminacy has consequences when trying to identify appropriate KBAs. Several different criteria can be used. For example, the sharing of data between processes, and so the amount of communication if the processes were separated; the balance of functionality between, or reasonable size of, processes (sometimes based on the size of the PSPEC in the requirements specification); the natural concurrency in a problem; a resemblance to a standard parallel architecture (for example, pipelining). These criteria do not all share the same level of abstraction. For example, communications concerns are concrete, and may conflict with a "logical" architecture design based on a natural concurrent decomposition of the application.

In this example, the reasoning about KBA identification went as follows.

1: The best (and easiest) metric to use is that of datastore usage. Diagrams and PSPECs can indicate the balance of datastore read operations (input flows) to datastore write operations (output flows); the former are to be considered more expensive.

2: Internal datastore usage is considered first. The group {2.5.1, 2.5.2, 2.5.3} share the Last Command data store, and {2.5.6, 2.5.7} share Time in Sector.

3: A criterion of external datastore usage is then employed. The group {2.5.4, 2.5.6, 2.5.7} are linked by the use of the Flight Plans datastore flow. (The Aircraft Positions flow does not come from a datastore). Adding in this metric yields the groupings: {2.5.1, 2.5.2, 2.5.3} and {2.5.4, 2.5.6, 2.5.7}.

4: This leaves the problem of allocating of 2.5.5; it can either be a separate KBA or be amalgamated with one of the other two. It is too small to be a KBA, so one has to decide which of the others to add it to.

5: Looking at the use of auxiliary functions in PSPECs, it is found that 2.5.5 has the use of Course_Diff in common with 2.5.1 and 2.5.2. A natural solution is to put it with these. There are now two KBAs corresponding to the requirements model of process 2.5: KBA2_5.1 = {2.5.1, 2.5.2, 2.5.3, 2.5.5} and KBA2_5.2 = {2.5.4, 2.5.6, 2.5.7}.

6: Confirmation of this decision is sought by examining the size of the two KBAs, based on the total size of the PSPEC text of the grouped processes. This reveals KBA2_5.1 to be about four and a half pages, with KBA2_5.2 about four pages. It was therefore concluded that this identification of KBAs would work reasonably well.

The place of these two KBAs in the final architecture of the ATC workstation is shown in figure 7. They appear as Monitor-Command and Monitor-Flight. (Note that, to ease the drawing task, the same Data Base appears many times).

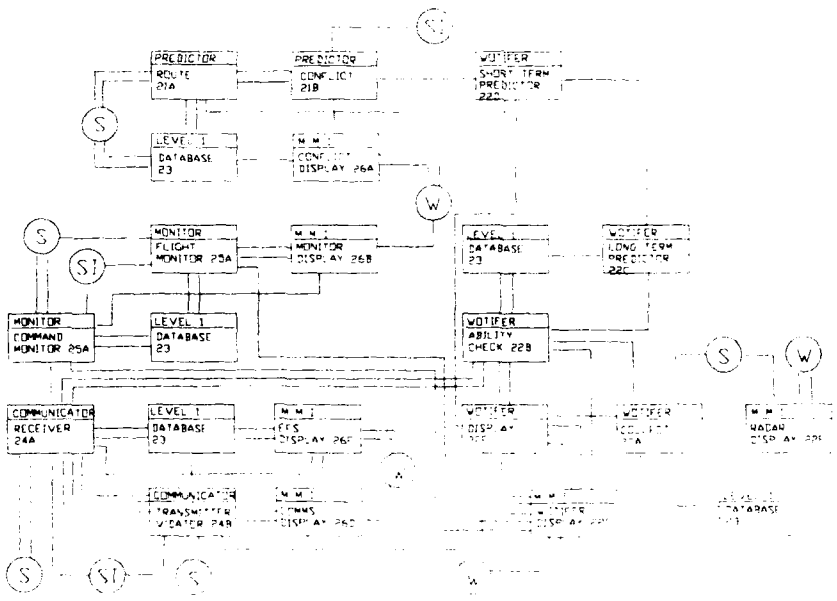


Fig. 7 Architecture of the ATC Workstation

Implementation. The architecture design determines how many Muse processes will be needed, since each Knowledge Based Agent is assigned a corresponding Muse process. (Obviously, the availability of machines and memory capacity can make certain decisions unrealisable). A typical Muse application would exploit the following elements. Knowledge sources are scheduled on an agenda when their rulesets are notified of changes to objects in various working memory areas ("databases"). Some of these databases are associated with knowledge sources and rulesets, some are stand alone ("notice boards"). Procedure calls and message-passing can supplement the work of rules, or can be used independently. A set of data channel objects capture external information from processes linked to one another by socket connections. Streams to and from these processes are objects to which write and read messages can be sent. Each Muse program is a regarded as an object that contains all the application structure in its slots and methods.

4.3 Dynamic Tactical Planning

The Dynamical Tactical Planning application (DTP) was chosen as the second driver application for a number of reasons. The scope of the method includes guidance on the issues such as cooperation that need to be addressed in developing distributed intelligent systems, but the requirement for these in the ATC system was minimal. In order to support the further development of the method and demonstrate its versatility, therefore, it was important for the second application to have a clear requirement for cooperation between agents.

This application consists of two teams of three autonomous agents manoeuvering within a bounded hexagonal grid. The agents have two principal attributes: Ammunition available and sustainable damage.

The arena of combat consists of a hexagonal grid with a home base for each aircraft in each of the six corners. The goal of each aircraft is to maximise damage inflicted on an enemy while minimising damage to itself. Each move consists of three phases which in turn consist of a possible movement and possible firings. The planning cycle for each aircraftaddresses the next three actions, where all three actions for each aircraft are taken together.

Although the aircraft agents were autonomous, for the most part their outward behaviour was that of a concerted team effort.

5 Conclusions

The objective of the CONSENSUS project has been the production of a method for the development of large real-time concurrent knowledge based applications. The scope of the method encompasses the complete product life-cycle including: functional analysis, the design approach, detailed design, implementation, verification, validation, testing, maintenance and support.

The method has been under continuous development, and refinement, throughout the duration of the project, with team members, distributed over three sites, occupying roles of both developers and users of the method. The requirements modelling phase of the method was considered to be the most effective component of the method, based on proven techniques for large scale development.

The method supports architectural design through a set of guidelines detailing how the processes and data stores of the requirements model should be allocated to the knowledge based agents, the principal architectural components of the design, while taking into account any system resource limitations. These guidelines were employed extensively and effectively on both demonstrator applications.

One of the end products of the design process is a set of Process Specifications (PSPECS). The PSPECS represent the functionality of the system and are usually expressed as structured English or truth tables etc.. The mapping of PSPECS onto the knowledge sources was performed as laid down in the method guidelines. The correspondence between the data flows of the requirements model and the architecture interface flows of the architectural design, together with the neutrality of functional decomposition ensures that the PSPECs transform naturally onto the structure required for knowledge sources within an agent. The use of formal knowledge acquisition tools, as part of the requirements analysis, is likely to help this process.

An effective approach to the development of hierarchical designs is based on using the notion of an abstract high level knowledge based agent, which has little or no functionality of itself, but which defines a decomposition into a number of other lower level knowledge based agents. This was exemplified by the architecture selected for the second application, where abstract knowledge based agents were used to provide conceptually simplified routing of inter-agent messages.

Accurate, architectural interface flow descriptions were of considerable benefit to the designers of the demonstrator, providing the required interface specification between the many processes used to implement the knowledge based agents of the application. The flow descriptions of the architectural design provided a base from which verification could be performed. In particular, the flow descriptions provided strong support for the construction of test harnesses for agent testing.

The documentation standards for the method, together with their associated document notation, are an important component of the CONSENSUS method. The benefits of applying such standards were demonstrated by the ease with which the development team were able to generate, share and review unambiguous details of both specification and design. The quality of documentation was the key to the support of the multi-site development of the demonstrator applications. In particular the documentation standards were considered to be a major contribution to the ease of development of the second application.

The method calls for the requirements modelling and architectural design phases to be carried out as concurrent activities. This was reflected in the structure defined for the design level documents, which required the parallel exposition of requirement modelling and architectural design, within the same design document. This approach was considered to provide exceptional benefits, and vindicated its adoption as part of the documentation standards for the method.

Support for the design of large scale systems is provided directly through the process of functional decomposition, which was applied with success to both demonstrator applications. The ability to generate a hierarchical architecture that has a correspondence with the process decomposition of the requirements analysis takes support for large scale systems even closer to implementation.

Knowledge based agents are the basic building blocks of the architectural design. This design approach leads to a design that is free from deadlock, and ensures consistency within the system while being able to meet real-time constraints, thereby providing the necessary support for the construction of continuous real-time applications.

The requirements model helps identify where concurrency in an application exists, and thereby enables concurrent designs to be created. However, the method does not directly support the exploitation of available concurrent computing resources where concurrency has not been identified through the modeling of requirements.

Functional decomposition was found to be good at highlighting where more knowledge about the system was required. The method provides no direct support for knowledge elicitation, limiting itself to defining a blackboard structure for the application's architectural elements in terms of knowledge based agents, where the blackboard is a generic platform supporting pattern-directed inference, that can employ a number of representation schemes. The application knowledge is represented in the PSPECS of the requirements model which are transformed into the behaviour of the knowledge sources within an agent.

In conclusion, the CONSENSUS method has been used in the construction of two applications, and has been refined in the light of the experience that has been gained. The project concluded at the end of February 1993 and the final version of the method will be available from the end of March [1].

References

1 M. Bateman: The CONSENSUS Method Final Report (D17), available from the authors

2 M. Bell & B. Clark B: Research into Air Traffic Tools(RATT). Final Report No. C22779-FR-001a, Cambridge Consultants Limited

3 A. Bond & L. Gasser: An Analysis of Problems and Research in DAI. In A. Bond A. & L. Gasser (Eds.): Readings in Distributed Artificial Intelligence. Morgan Kaufmann 1988

4 Muse: System Guide and Reference Guide. Cambridge Consultants Limited 1988

Script and Frame: Mixed Natural Language Understanding System with Default Theory

Honghua Gan*
Department of Computer Science
University of Exeter
Exeter, EX4 4PT UK
Email:hga@dcs.exeter.ac.uk

Abstract

Minsky's frame [7] and Schank's script [10] are two leading representation languages in the natural language understanding system for understanding a story. Very different inference mechanisms are embedded in the two representations: property *inheritance along taxonomic* structure in the frame system vs. causal-effect connectivity among a sequence of events in the script. This paper attempts to merge them into a mixed understanding system, specially via default logic to formalize its inference process and create a causalized default theory. The idea is to retain the frame system as a basic structure for events, and then to organize a script specifying the normal way of events happening in the specific situation via connecting some concerned lower level frames with causal relationship.

1 Introduction

One of the aims of Artificial Intelligence is to make computers understand human's natural language. Much better than the current machines, human beings can use the so-called common-sense knowledge in their everyday reasoning to help understand stories, make decisions, and take actions. In order to assign similar abilities to computers, many formal or informal more powerful inference mechanisms have been presented [eg. 8,6; 7,10].

In understanding a story, there are two classical theories handily. For a descriptive story such as a news story [14], frame systems work well; for a sequence of events involving actions, scripts seem to be more suitable and efficient [10]. In both frame systems and scripts, there are defaults hidden in the explicit inference mechanisms. Property inheritance is a main feature of frame systems.

*The author is supported by Sino-British Friendship Scholarship Scheme (SBFSS) and Natural Science Foundation of China

It can be roughly specified that properties can be shared by the different frames and normally can be inherited by the lower level frames from the higher level frames along the taxonomic structure, unless the contrary is explicitly filled in the local slots. In many situations, the property inheritance process is defeasible, and it can be formalized by the default theory [1,2]. On the other hand, scripts enjoy a causal-effect chain among the event sequence of the story in the specific situations. In fact, the script specifies a normal way of what has taken place based on commonly accepted experience and supposes these events have really happened unless the story has told something different. This normal supposition of thing's taken place is defeasible, too, and much information should be gap-filled in the understanding process. An initial attempt aiming at formalizing scripts and script-based understanding in a default theory based on default logic has been tried in [5].

In this paper, further understanding systems are considered. In the basis of investigating frames and scripts-based understanding systems respectively, a mixed representing mechanism is proposed and an attempt to formalize the mixed inference in a default theory is provided subsequently.

2 Representing Static Properties in Frames

Frame system is a leading knowledge representation language, as Minsky proposed, it could be compared to the dominated logic-based mechanisms. The essence of the theory is that "when one encounters a new situation (or make a substantial change in one's view of the present problem), one selects from memory a structure called a frame."[7] The basic features can be extracted as follows:

- A frame is a data structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party.
- For visual scene analysis, the different frames of a system describe the scene from different viewpoints, and the transformations between one frame and another represent the effects of moving from place to place. Different frames of a system share the same terminals.
- A frame's terminals are normally already filled with "default" assignments. Shared frames can inherit these default assignments from the higher level frames along the taxonomic structure.

Therefore, frames in essence are suited to represent static or descriptive properties from one view. For example, a natural language story, we suppose, contains a series of isolated events. Obviously, all descriptive properties of these events are easy to be represented in the frame systems. One simple example is to represent disaster events in the event frame [14] shown in Fig. 1.

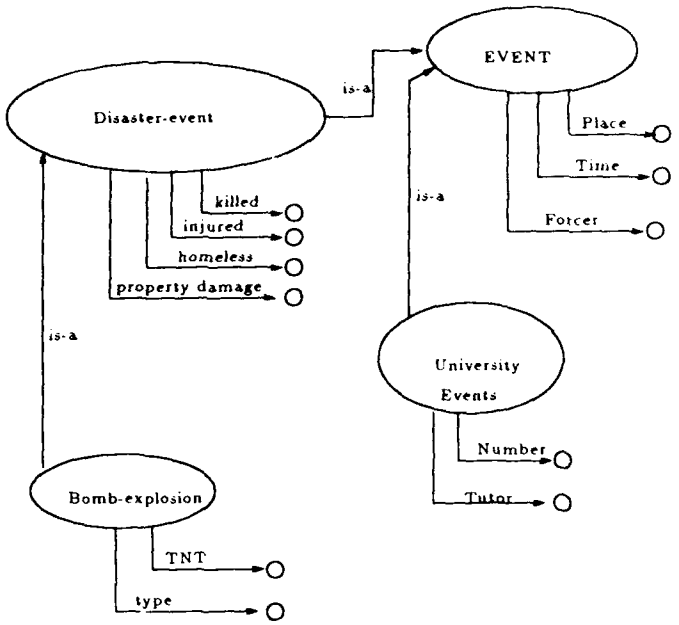


Figure 1: Disaster Events in A Frame System

It is easy for computers to understand such stories. When you ask questions about the information concerned with the disaster, the understanding system can correctly give answers through inheritance inference.

3 Normal Sequence of Actions in Scripts

In spite of obvious bias towards static properties, Minsky and some other researchers [7,4] tried to represent action-type events in their frame systems, which were called "scenarios frames". But they have not worked properly. The similar attempt is Schank's script, which provides a supposed event's process in detail in the specific situations. From the help of scripts, the understanding system easily learns causal relations between isolated events given by the story.

The characteristics of the script-based understanding systems can be concluded as follows:

- The script is used for representing a well-known common situation, which is typically commonsense knowledge. The script is heavily based on past commonly-accepted experience, reflecting the normal way of something happening.

- There are causal relations between the adjoint events in the script. The understanding system gives a causal explanation for the isolated events in the story via connecting them with events in the script.
- There are some default slots in the script, which expect to be filled after the story is stepwise inputed. After filled, they can be normally referenced identically.

Much different from frames, scripts depend upon causal relationship passing the useful information between events. Namely, the inference mechanism in the script-based understanding system is a causalized default reasoning, which supposes something happening between ill-connected events from the story and fills in the gaps to make them well-connected.

It is easy to see that the script itself is defeasible knowledge specifying the normal way of actions. But when something different is explicitly mentioned in the real story coming out stepwise, the events in the script have to be overridden by these facts.

In a stepwise story understanding system, we suppose the story consists of a sequence of events $e_1, e_2, \ldots, e_n$, which are ill-connected. We must find normally taking place events from the script to fill in the gaps to make them well-connected. Suppose the script contains the causally connected events $se_1, se_2, \ldots, se_k$, the understanding explanation is like this

$$e_1[= se_1, se_2, \ldots, se_{i_1} =]e_2[= se_{i_1} \ldots se_{i_2} =] \ldots e_n,$$

where the events between pair e_i, e_{i+1} can be null.

4 Merging Scripts and Frames

There are some common points between frames and scripts, one of the most distinguished is that there are some slots in both frames and scripts to represent some kinds of static properties of the events. We retain frame structure as a basic framework of our mixed system so that this knowledge will be only specified in the frames. In fact, there are two kinds of defaults in the script. One is object-oriented, ie. slots denoting actors, objects etc. as mentioned above. The other is sequence of actions itself, which is hard to embed in the frame system. Compared with frames, the second kind of defaults in the script specifies dynamic properties of event's sequence. Our task is to create a mechanism to represent such dynamic properties of actions in the frame structure. Namely, we should capture scripts in the frame structure in some way.

As Schank proposed [9], actions in a typical story can be classified a small type set, called primitive acts. Primitive acts abstract action behaviors from the detailed situations and can be installed into a script in the specific case. The

problem is how to represent these primitive acts and their subsumed detailed actions in the frame systems. We recall that the frame can be also used for representing scenarios. Now we do not want the frames to represent the dynamic process, but representing static property of action-type knowledge. Namely, primitive acts are represented a kind of special frames, which actually are a set of actions. One action-type frame includes four slots:

*actor: ********(executive of the action)*
*action: ********(primitive acts if possible)*
*object: *******(the object of the action)*
*direction: *******(from and/or to of the action)*

We obtain a whole event frame system, which includes two categories of frames: one is for descriptive events and the other for action-type events. The action-type events specify dynamic change of event's sequence. From time to time, some of them are organized as scripts, with respect to one specific descriptive frame. So, in fact, we have two relationships in the frame systems: one is inheritance via is-a taxonomic links and the other causal-effect relation via causal connectivities in the scripts. Given a story step by step, if the event is only concerned with the static properties, it is not necessary to invoke the script, the inheritance mechanism is enough: fill in the slots or obtain the default values through inheritance paths; otherwise, the events are not well-connected between adjoints, the script must be activated to help understand why the events look like this. Normally, one event from the story can get all information from the is-a relation and causal relation in the script.

It should be notified that the inference granularities in the descriptive frame and in the action-type frames in the script are different. In the descriptive frame, the concerned slots are customers, waiters, cooks, menus, foods, cashiers etc. On the other hand, in the script, what is focused upon is about customer, waiter etc's behaviors! The customer John is not important in the whole story, something important in understanding is John's behaviors through the story: enter the gate, find a place to sit down, look up the menu, order a dish, eat the food, pay the check, and exit the gate.

Fortunately, it is easy to distinguish these two inference granularities. Normally, the static properties which are specified in the descriptive frame are passed by inheritance mechanism along the taxonomic structure. Conversely, dynamic properties which are specified in the script are supposed as really taken place based on causality. Given a stepwise story, one can fill the static properties in the slots of the descriptive frame and obtain causal explanation from change of slots of action-type frames in the script respectively. In the default theory below, the two default reasonings carry on in different layers, therefore there is no confusion at all.

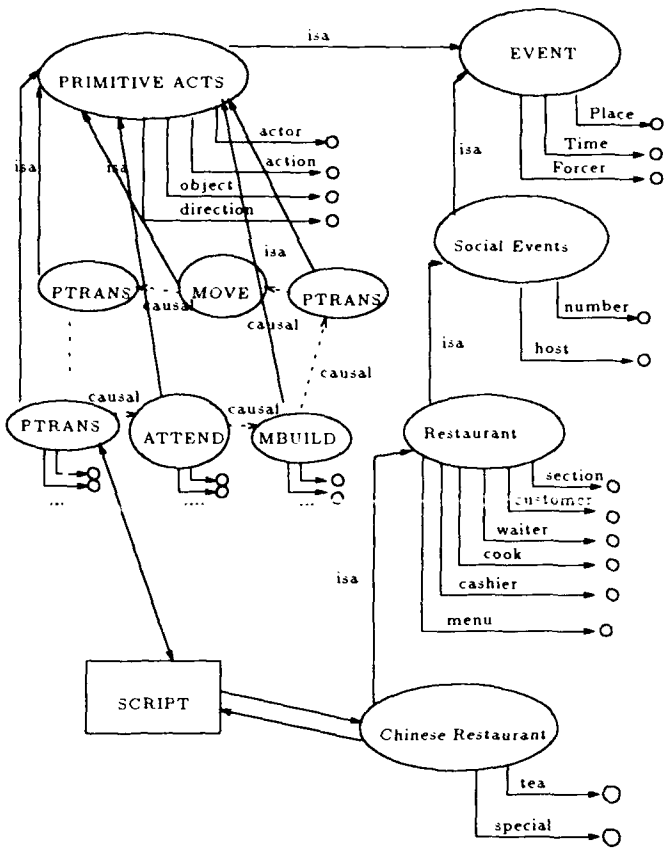


Figure 2: Mixed Representation for Restaurant

5 Formalizing with Default Theory

We suppose that we can create an extendable all-event frame system, which contains events that could take place at any time. As specified in the last section, two categories of frames exist in such a system: descriptive frames for static properties and action-type frames for change of events. Correspondingly, there are two distinguished inference mechanisms in the system. All descriptive frames take property inheritance along the is-a taxonomic structure vertically; and constrained by the descriptive frame, the associated script makes some action-type frames connected with causal relation reflecting change of events normally specifying the descriptive frame.

We try to apply default logic to merge two different inference mechanisms in a uniform formalism. Brewka [1] gave an excellent semantic explanation for inheritance inference mechanism in the frame system with McCarthy's circumscription. Similar considerations can be found in [2], with the other nonmonotonic reasoning formalisms. Reiter's default logic can give a simply clear default theory for a typical inheritance inference.

Suppose one frame structure in the frame system consists of a frame name, several slots and corresponding (default) values of the slots. The lower level frames can obtain the (default) values of slots from the higher level frames if there is no explicitly assignments given at the lower level frames for their relevant slots. Let's denote f, or $f_1, f_2, \ldots$ to frames, s, or $s_1, s_2, \ldots$ to slots of a frame, and v, or $v_1, v_2, \ldots$ to values of the slots. The inheritance inference can be formalized as follows:

- $Hold(f_1, s, v) : isa(f_2, f_1) \wedge Hold(f_2, s, v)/Hold(f_2, s, v)$ $(\delta_{D_1} \in D_1)$

Namely, for the frame f_2, if there is no (default) assignment for its slot s, then according to inheritance mechanism, along the is-a relation (ie. the frame f_2 is a sub-frame of the frame f_1) the value v of the slot s in the frame f_2 persists the corresponding one as in the higher frame f_1. Here the predicate $Hold(f, s, v)$ represents that there exists a slot s with the value v in the frame f, and the predicate $isa(f_1, f_2)$ reads that the frame f_1 is a sub-frame of the frame f_2.

There is something more complicated in formalizing the script. Clearly, inference in the script consisting of action-type frames is not directly dealing with the slots and their values of the activating descriptive frame. The inference granularities in the two layers are quite different: one is about the properties of the slot's values of the activating frame; the other the slot and value itself. Recall the four own slots of an action-type frame: *actor*, *action* (primitive acts), *object* and *direction*, the default theory must reflect *actor* and *object*'s behaviors under default *action* and *direction*, where the *actor* and *object* respond to the slot's value of the activating descriptive frame. The defaults seem like these:

- : $causal(fa_1, fa_2) \wedge Hold(fa_2, *actor, va)/Hold(fa_2, *actor, va)$ $(\delta^1_{D_2} \in D_2)$
- : $causal(fa_1, fa_2) \wedge Hold(fa_2, *action, va)/Hold(fa_2, *action, va)$ $(\delta^2_{D_2} \in D_2)$
- : $causal(fa_1, fa_2) \wedge Hold(fa_2, *object, va)/Hold(fa_2, *object, va)$ $(\delta^3_{D_2} \in D_2)$
- : $causal(fa_1, fa_2) \wedge Hold(fa_2, *direction, va)/Hold(fa_2, *direction, va)$ $(\delta^4_{D_2} \in D_2)$

Where, the predicate $causal(fa_1, fa_2)$ represents that there is a causal relation from the frame fa_1 to the frame fa_2, fa_i refers to an action-type frame. The informal explanation for the above defaults is that based on the causal chain in the script, the behaviors of the four slots (responding to the values of the relevant slots of the activating descriptive frame) are the normal process of the event's change unless the contrary is explicitly mentioned in the real story.

Suppose the real story is stepwise arriving, $\{e_1, e_2, \ldots, e_n\}$, which are facts. Now together with the defaults about traditional frame structure and scripts, we have a default theory $\Delta(D, W)$, with
$D = D1 \bigcup D2$
$W = \{e_1, e_2, \ldots, e_n\}$
Suppose the fact event e_1 comes, the descriptive frame f is invoked, and all its slots will be filled in. Then the associated script (if any) will be activated, which tries to connect the substantial events $e_2, \ldots, e_n$ as possible as it could.

6 Examples

There are some classical examples in the understanding system, such as understanding a news story [14], about going into a restaurant [10], and about shopping [4]. We try to analyze two of them with our default theory for the mixed understanding system.

Example 1 :

Today an extremely serious earthquake of magnitude 8.5 hit Lower Slabovia killing 25 people and causing $500,000,000 in damage. The President of Lower Slabovia said the hard hit area near the Sadie Hawkins fault has been a danger zone for years.

There is no script at all for understanding this news story, because all information is about static properties of the frame *earthquake*. Nothing is provided about a sequence of actions normally happened in the earthquake. The default theory for that is $\Delta(D, W)$, with
$W = E$ (one event — earthquake)
$D = D1$ (only concerns with the descriptive frames)
Obviously the slots of the frame *earthquake* have the following values:
place — Lower-slabovia;
time — Today;
forcer — elements;
killed — 25;
*injured — ***;*
property-damage — 500 millions;
magnitude — 8.5;
fault — Sadie Hawkins.

Example 2 :

John went into a Chinese Restaurant. He asked for a dish of lobster. He paid the check and left.

Here, the story comes with "John's entering the Chinese Restaurant", the descriptive frame $Chinese - Restaurant$ is invoked. The slots of the frame are filled in as possible as it could, and then the associated script is activated. Now, the slot *customer = John*, and according to causal relation in the action-type frames of the script "Restaurant", the default theory produces the event sequence such as $fa_1, fa_2, \ldots$. The second events in the story is arriving, e_2, again, the understanding system fills in the corresponding slot of the descriptive frame and then matchs the fact with some action-type frames in the script (because if the mismatch happens, it actually means that there is some contrary explicitly mentioned in the story, and the default has been blocked).

Finally, the frame has got default values about its slots:
customer — John;
*waiter — ***;*
*cook — ***;*
menu — on the table;
*sitting-at-table — number ***;*
food — lobster;
check — payable;
*tips — ****
etc.

Meanwhile, the understanding system gives the causal explanation what has happened about *John* eating in the Chinese Restaurant:
$John * PTRANS * \stackrel{causal}{\Rightarrow} John * ATTEND * \stackrel{causal}{\Rightarrow} John * MBUILD * \stackrel{causal}{\Rightarrow} \ldots\ldots$.

7 Some Related Work and Discussion

Minsky gave a birthday example [7], which probably was the earliest attempt to do dynamic process of events in the frame. But the drawback is that there is no causality between the frames. Only depending upon the detailed frame analysis, there is no way to make it clear. Similarly, Charniak's system [4] did not give a clear causal explanation for frames although he claimed that his system seemed to be no significant differences from scripts. Obviously, the very important different inference mechanisms between the causal relation in the script and inheritance in the frame were ignored.

Our mixed understanding system based on a default theory should be better than both of them. In practical, nevertheless, it is difficult to organize action-type frames to be scripts, and sometimes it is subtle to keep the trace for the descriptive frame and its action-type frames in its associated script, because

some slot's values are provided by the late events in the story, which may lead to a conflict with the script.

References

1. G. Brewka: The logic of Inheritance in Frame systems. In: Proceedings of IJCAI-87, Vol 1, pp. 483-488, Milan, Italy, 1987

2. G. Brewka: Nonmonotonic Reasoning: Logical Foundations of Commonsense. Cambridge University Press 1991

3. A. W. Burks: Chance, Cause, Reason: An Inquiry into the Nature of Scientific Evidence. The University of Chicago Press 1977

4. E. Charniak: Inference and Knowledge Part I, II. In: E. Charniak and Y. Wilks (eds.): Computational Semantics: an introduction to artificial intelligence and natural language comprehension. Oxford: North Holland, 1976

5. H. Gan: Formalizing Scripts with Default Theory. Technical Report 248, University of Exeter, Department of Computer Science, UK, 1992

6. J. McCarthy: Circumscription — A Form of Nonmonotonic Reasoning. Artificial Intelligence 13:27-39 (1980)

7. M. Minsky: A Framework for Representing Knowledge. In: R. J. Brachman and H. J. Levesque (eds.): Readings in Knowledge Representation. pp 245-262, Morgan Kaufmann, 1985

8. R. Reiter: A Logic for Default Reasoning. Artificial Intelligence 13:81-132 (1980)

9. R. C. Schank: Conceptual Information Processing. North-Holland Publishing Company 1975

10. R. C. Schank and R. P. Abelson: Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures. The Artificial Intelligence Series. Lawrence Erlbaum Associates Publishers 1977

11. R. C. Schank and C. K. Riesbeck: Inside Computer Understanding: Five Programs Plus Miniatures. Lawrence Erlbaum Associates, Inc. 1981

12. R. C. Schank: Reading and Understanding: Teaching from the Perspective of Artificial Intelligence. Lawrence Erlbaum Associates, Inc. 1982

13. R. C. Schank: Dynamic Memory: a theory of reminding and learning in computers and people. Cambridge University Press 1982

14. P. H. Winston: Artificial Intelligence. Addison-Wesley, 2nd edition, 1984

Constructive Matching Methodology: *Formally Creative* or *Intelligent* Inductive Theorem Proving?

Marta Fraňová, Yves Kodratoff, Martine Gross

CNRS & Université Paris Sud, LRI, Bât. 490, 91405 Orsay, France

Abstract. In this paper we explain why, and in what sense, the methodology for inductive theorem proving (ITP) we develop is creative and we explain why our methodology cannot be said to be "intelligent", as a human could be, and nevertheless it is suitable for a user-independent automatization of ITP.

Introduction

We have been developing for almost a decade a methodology, called *Constructive Matching* (*CM*), to automatize inductive theorem proving (ITP) suitable also for automatic construction of programs (ACP). The goal of this paper is to stress upon a non-usual character of our methodology, we shall call formal creativity (FC) and describe it later, as well as to explain its origin in our methodology. The reader will understand that FC expresses a behavior of an automated system that is similar to a standard routine work of a mathematician. Let us recall how it looks like.

Suppose that a mathematician is proving a theorem called THEOREM. During his proof he realizes that in order to perform an operation, say OPERATION, he needs to prove a lemma, say LEMMA. He starts to prove this lemma. At some step, he realizes that LEMMA is not true in general, but only in particular cases. He realizes that, in fact, he can use these particular cases to generate a new lemma LEMMA2, which allows for the performance of OPERATION as well. Thus, the mathematician knows how to analyze his failure for LEMMA. He knows that this failure is not significant, because he succeeded in finding another solution for performing the OPERATION.

We speak here of a standard routine work, since the most interesting proofs found by a mathematician are based on a clever trick, or on a suitable combination of previously proven lemmas seemingly having nothing to do with the given theorem. We shall mention also systems that behave in this clever way.

As a behavior, FC concerns also user-independence of automated systems, and thus we shall say few words also on this topic. Finally, we shall specify the place of *CM* in the automatization of ITP and ACP. Later we shall mention some technical papers describing the *CM*-methods and techniques, as well as the differences of our methods and techniques with the methods and techniques of other approaches in ITP and ACP. In this paper we choose not to present technical details of *CM* so that it can be understood by anyone with a slight experience in mathematics or logic.

The paper has the following structure. Section 1 recalls the goal of ITP and links it to ACP. Section 2 introduces the notion of formal creativity in the automatization of ITP as an antonym to human creativity, and links the mentioned types of creativity and types of systems with regard to their dependency on a user. Section 3 describes how we characterize existing approaches to ITP and ACP with respect to these two types of creativity. In section 4 we explain the origin of FC in our methodology examining slightly a possibility to transfer our FC to other existing approaches. We mention also the present implementation of CM, report some interesting results, and describe topics for future research.

1 Inductive Theorem Proving and Programs Construction

A *formal proof of a theorem* T in a *formal theory* $\mathcal{T}$ (theory for short) is an ordered sequence $S = \{ A_1, A_2, \ldots, A_n, T \}$ of formulae such that each formula is either an *instance of* an *axiom* of $\mathcal{T}$ or the result obtained from the preceding formulae by using an *inference rule* of $\mathcal{T}$. The problem of *theorem proving* consists of finding a formal proof in $\mathcal{T}$ for a given theorem T.

In *inductive theorem proving*, we deal with universally quantified theorems, i.e., formulae of the form " For any object, say x, the property F holds". This is usually expressed symbolically as $\forall x\ F(x)$. For these theorems we use a special inference rule, called the *induction principle*, in order to discard the universal quantifier whenever our objects, represented here by the variable x, belong to a well-founded domain. For simplicity, in this paper we restrict ourselves only to nonnegative integers, $\mathcal{NAT}$, which are familiar to any reader. Of course, there are other well-founded domains to which the induction principle can be applied, and such domains are treated in our technical papers. In $\mathcal{NAT}$, the induction principle is represented by the scheme $F(0),\ F(n) \Rightarrow F(n+1)\ \vdash\ \forall x\ F(x)$. This induction scheme says that in order to prove the formula $\forall x\ F(x)$ for natural numbers, it is sufficient to prove the formulae F(0) and $F(n) \Rightarrow F(n+1)$ for an arbitrary natural number n.

The induction principle changes somewhat the notion of a formal proof of a theorem T. Consider a theorem $\forall x\ F(x)$, where x belongs to nonnegative integers and F is a property we want to prove. A *formal inductive proof* consists of two ordered sequences $S_1=\{C_1, C_2, \ldots, C_n, F(0)\}$, $S_2=\{B_1, B_2, \ldots, B_k, F(n+1)\}$ of formulae. Each formula is an instance of an axiom of $\mathcal{T}$, or (in S_2) an instance of the formula F(n) called the *induction hypothesis*, or the result obtained from the preceding formulae by using an inference rule of $\mathcal{T}$. We shall call *inductive decomposition* the transformation of the set {axioms, $\forall x\ F(x)$ } characterizing the problem to prove $\forall x$ F(x) from a set of axioms, into $S'_1 = \{$axioms, F(0)$\}$, called the *base step*, and $S'_2 = \{$axioms, F(n), F(n+1)$\}$, called the *induction step*. These sets express the task to prove the last formula of the set from the previous ones.

The goal of ITP is to (i) perform the induction decomposition, i.e., specify correctly the last formulae of sequences S'_1 and S'_2, as well as the form of the induction hypothesis. This amounts to generating a correct induction scheme for a given domain and a theorem. The induction decomposition for general well-founded domains is not so trivial as it could seem from the above decomposition in $\mathcal{NAT}$. (ii) from S'_1 and S'_2 provide corresponding sequences S_1 and S_2 for a given theorem.

The first task differentiates ITP from classical theorem proving. While proving theorems by induction has been long recognized as a useful way of solving some mathematical problems [29], the interest in mechanizing ITP became strongly evident when people realized its applicability to *programs verification* and to construction of programs from formal specifications. Let us recall first how ITP applies to *program construction* in case when the desired program f is specified formally by something similar to: *for any given input x verifying the input condition P, the output z has to be such that the input-output relation Q(x,z) holds*. An interesting feature of ITP is that if we are able to prove by induction a theorem of the form

$$\forall x\ \{ P(x) \Rightarrow \exists z\ Q(x,z) \}, \qquad \text{(ST)}$$

called usually a *specification theorem*, then a by-product of such a proof is a program f verifying the formula

$$\forall x\ \{ P(x) \Rightarrow Q(x,f(x)) \}. \qquad \text{(VFP)}$$

The task to construct programs specified by (ST) is called *Program Synthesis* (PS). The formula (VFP) allows now to understand the role of ITP for program verification. Once we have a program f and a formal specification, by ITP we can check if our program verifies the given specification.

By this we have completed a rough description of the goal and use of ITP. Let us say few words on the "art" to prove theorems by induction. Mathematicians, logicians and many computer scientists usually do know how to prove a theorem by induction when they need it. This is quite interesting since, to our best knowledge, they do not learn at school *how* to prove theorems by induction. Of course, they have seen in school some inductive proofs, but they did not have a course presenting *mechanisms* for proving theorems by induction allowing, just on the basis of these mechanisms and not depending on acquired experience, to prove *any* theorem that professor himself is able to prove. In other words, presently, people learn how to prove theorems by induction only through experience. This indicates that there may be some difficulties in automating ITP, i.e., in implementing a system able to prove theorems by induction. In the next section we shall be interested in the behavior of systems implementing ITP.

2 Formal Versus Human Creativity

2.1 Formal Creativity in the Automatization of ITP

The notion of formal creativity may be pictured as a behavior of an ordinary (concerning the intelligence), but diligent (concerning the work) student. Such a student is able to prove only theorems for proofs of which "routines" are learned at school. In order to prove a theorem, he just follows these routines. In order to be formally creative, an ITP system must implement these routines. In other words, formal creativity in ITP comprises the ability to provide correct solutions (i.e., as specified in section 1, induction decomposition plus sequences S_1 and S_2). It also includes the ability to *understand* and *analyze failures*, as well as the ability to *propose new solutions for recovery* based on such analysis. This reminds what we have described, in introduction, as a standard, routine work of a mathematician.

We have mentioned above that people learn how to prove theorems by induction through experience. This hints somewhat at the absence of an ITP-methodology allowing to prove non-trivial theorems without understanding the semantic behind these theorems. In other words, to learn how to prove theorems by induction means that one is able to treat the functions and predicates occurring in a theorem to prove as symbols, and an ITP methodology describes procedurally how to manipulate these symbols -- and the definitions corresponding to these symbols -- so that a formal proof of the theorem is obtained. The reader may realize or understand better the difficulty of syntactical proof manipulations by preparing the following exercise.
1.- take a non-trivial theorem, for instance Theorem 18 from [30], 2.- use abstract function and predicate names, such as f_1, f_2, ... and P_1, P_2, ... instead of those used by Skolem, 3.- define f_1, f_2, ... and P_1, P_2, ... as Skolem does, and then simply prove the resulting formula by induction -- without knowing about what this theorem speaks of, without knowing what f_1, f_2, ... and P_1, P_2, ... mean. In case of a success, if still not convinced of difficulties of the automatization of ITP, the reader should try then to prove the prime factorization theorem, i.e., the problem of finding a decomposition of a given natural number into its prime factors, using axioms from

[22]. In case of a success, the reader may compare both proofs. We claim that there will be little similarity between them or even between parts of them.

2.2 Human Creativity in the Automatization of ITP

Human creativity, concerning ITP, may be pictured as a behavior of a very bright, almost ingenious student. Such a student is able to provide almost unexpected and highly interesting proofs showing he masters not only ITP, as a formal technique, but also the domain in which he proves theorems. An implementation of a human creative system is able do the same as this ingenious student.

Obviously, a human creative (HC) system is more appealing than a formally creative (FC) system. Therefore, if we say that our methodology tends to be formally creative, while other existing approaches are nearer to human creative systems, the reader may just want to stop to read now. Well, there is a small drawback in human creativity of existing approaches. We shall tell more about it in section 3 and Conclusion. Presently, let us mention only that this drawback manifests itself by the fact that until now no one has proposed an ITP system behaving always, when it comes to the results, as an ingenious mathematician. It will become clear later that, presently, ITP systems providing interesting proofs require in some way a guidance of a user. Therefore, we allow ourselves to say that an automated system depends on *human creativity, if the user has to guide, in some sense, the system.*

2.3 Some More Evidences on User-independent Systems

It is obvious that if a system is really automated, a *user does not need to know how this system works*. It is also obvious that an automated system has to be able to *return the output expected* by the user. The problem of developing a user-independent system providing interesting proofs is a problem of implementing creativity, which, as we all know now, is far of being achieved easily. However, let us suppose that we decompose the wanted task to two tasks: 1. Develop first a completely automated system able to prove theorems by induction. (We do not forget here the undecidability of theorem proving. We speak here of a system which will do "its best".) This is the task of implementing "formal creativity", or, in other words, finding a systematic way to prove theorems by induction. If this task is achieved, then 2. Study how creativity can be introduced into this system.

We believe that a solution to the first task may be an FC system providing standard outputs. This means that an FC system will provide solutions also to problems that are not known by the user. Standard outputs mean that if the user knows a solution, an FC system will generally not provide the solution identical to that of the user [11].

Who may now be interested in an FC system? Students might. Interesting proofs are not always necessary for students. A professor may judge that his student is not very clever, since he has not discovered a tricky proof. He cannot deny that the student has proved the given theorem. Therefore, if students could learn a systematic way of proving theorems by induction, they would have more time to focus on other important things. For the same reason an FC system could be of some use for computer scientists.

3 Representatives of Research on Automatization of ITP

In section 2.1 we have given the reader a possibility to check by his own efforts that the automatization, user dependent or not, of ITP is not a simple problem. We have also spent a considerable time on a characterization of FC and HC systems, and we have mentioned that while the goal of our approach is to obtain an FC system, other existing approaches rather lead towards HC systems. However, we have not named these "other" approaches in ITP, and thus the reader might feel that our division of the existing approaches into ours and others comes simply from the fact that we do not know sufficiently existing work on automatization of ITP. Here are some hints at the reasons of insisting upon our division.

Let us start first with a natural division of ITP, such as it actually is accepted in Computer Science. To explain the origin of this division, as we see it, we have to recall first that specification theorems contain existential quantifiers. The difficulty with this kind of theorems lies in the fact that PS requires constructive proofs for ST while for purely universally quantified theorems there is no need for a proof to be constructive. Automatization of proving theorems containing universal quantifiers only is already a difficult problem. The requirement of constructive proofs for specification theorems seems to make the problem even more difficult. Therefore, it is not surprising that investigators divided their effort in building inductive theorem provers.

On one side, purely universally quantified theorems are treated by what is now called *automated ITP*. Let us mention here only the representatives of different approaches in this stream (In [20] we give the most detailed description of existing approaches). [3] is, undeniably, the representative of all the work considered as an improvement or extension of the approach developed by Boyer and Moore. We choose [6] to represent building constructivist or intuitionistic formalisms suitable for representing inductive proofs, and [25] is for us a representative of the approach using rewrite systems (or induction-less approach, as it is sometimes called) to prove theorems by induction.

On the other hand, specification theorems are treated by *program synthesis* from formal specifications. Manna and Waldinger [26] performed the pioneering work in PS. They are usually considered as representatives of deductive approach to ACP, i.e., the approach that constructs inductive proofs for specification theorems. [11] presents representatives of particular approaches in this deductive one.

Thus, this classical division in two streams, with respect to the difficulty of automating ITP is not surprising. What, from a naive point of view, is surprising, is that the division is so strong that, as one could conclude from literature if paying attention to it, the two streams exist separately in the sense that people working on universally quantified formulas develop methods that lead to subproblems that are purely quantified only, while PS approaches limit mostly themselves to generating subproblems that are of the form ST. In consequence, the mechanisms developed in these two streams have a common feature: they are not very suitable for theorems in the other stream, even if there are attempts to use some common computational techniques in one or other streams (see more in [11]).

With respect to this classical division, our approach has its origin [18] in the second stream, and falls under the class represented by [26]. However, very soon it became clear [19] that focusing on proving specification theorems rather then synthesizing programs makes our approach to fall under the scope of the first stream as well. The previous sections allow us to name the main feature of our approach:

formal creativity. We may now say that our work is a representative of the research on formal creative implementation of ITP.

4 *Constructive Matching*: A *Formally Creative* Methodology for ITP

4.1 Goal

The goal of $\mathcal{CM}$ is to capture conceptually problems in ITP and provide procedural solutions to these problems. More exactly, the goal of $\mathcal{CM}$ is to determine what are the problems innate to ITP, what are the problems linked to a user-independent implementation of ITP and how to solve these two types of problems so that the final implementation of our approach provides a user-independent system. We have previously explained that this goal means nothing but the goal to develop a methodology for ITP.

4.2 Restrictions

In the extended version of this paper [11] we present the basic restrictions, and in [10] more details are given. Let us assure the reader that the restrictions of $\mathcal{CM}$ do not make trivial the field of its applicability (as it is illustrated in section 4.5), and no ITP methodology has been developed for this particular field.

4.3 Formal Creativity of $\mathcal{CM}$ - Where it Comes From?

To answer "Why $\mathcal{CM}$ is formally creative?" means that we explain why $\mathcal{CM}$ provides an answer to two problems, presently recognized as major ones, concerning the automatization of ITP (cf. The call for AAAI-93-workshop on ITP): (1) How to generate suitable induction hypotheses' schemata?, (2) How to generate missing lemmas?

In $\mathcal{CM}$ these two problems are intertwined in the sense that our answer to the second question gives us the possibility to give almost a trivial solution to the first one. The solution to the first problem is based simply on the particular form of the structural induction principle we use (see [19] or [24]). When none of the generated in advance induction schemes applies for a particular theorem, the answer to (2) helps to find missing schemes.

If our solution for (1) is so simple, one may wonder where are the difficulties for other approaches in accepting our forms of induction hypotheses. Because of the classical division between ITP and PS approaches, we describe apart the main difficulties. Roughly speaking, ITP approaches cannot accept our solution to (1) because induction schemes generated in $\mathcal{CM}$ may contain universal quantifiers. The universal quantifiers in induction hypotheses may lead to a subproblem containing an existential quantifier. These, as mentioned above, are out of scope of ITP approaches. Let us suppose now that ITP approaches will decide to accept these universally quantified hypotheses accepting to use PS approaches when they meet such an existentially quantified subproblem. Unfortunately, this alone will not be sufficient to give ITP approaches a complete answer to (1), since, as we have mentioned already, the set of the $\mathcal{CM}$ induction schemes may be not complete. Therefore, ITP approaches would need to complete in some way the schemes proposed by $\mathcal{CM}$. We suspect highly that their solution will necessarily be similar to that of $\mathcal{CM}$, which relies on linking this problem to (2), in particular, to the problem of proving implications, i.e., formulae of the form $A \Rightarrow B$ (see more in [11]). In other words, we think that as

soon as ITP approaches will extend their scope to existential quantifiers and as soon as they will be able to prove implications, they will have their solution to (1).

Now, PS approaches may have objections towards our solution for (1), simply because our solution does not guarantee that programs -- synthesized while proving a specification theorem and using $\mathcal{CM}$-induction hypotheses -- will be as efficient as one programmer would be able to obtain. Once a standard proof leading to a standard program is obtained, $\mathcal{CM}$ can attempt to find a better proof in the way described in [15]. However, for lack of time and financial means, "better proofs" are out of scope of our present research, simply because we want first to complete our research on "standard proofs". Nevertheless, as it has already been mentioned, proofs leading to *the* most efficient programs are presently matter of human creativity.

Concerning $\mathcal{CM}$ solution to (2) we attribute our present success to the mechanism, the so-called $\mathcal{CM}$-formula construction ($\mathcal{CMC}$ for short), we have developed for proving atomic formulae [19]. Of course, $\mathcal{CM}$ applies not only to proving atomic formulae, however, as we see it now, our "luck" started by $\mathcal{CMC}$.

To understand why a mechanism for proving atomic formulae is fundamental for the automatization of ITP would mean to go back to logical connectives, de Morgan laws (see [20]), finally realizing that once we know how to prove atomic formulas and certain implications, we have almost solved the problem (see also [16]). Let us suppose now that the reader admits our stress upon atomic formulae, and let us explain where our mechanism differs from mechanisms used in other ITP and PS approaches. In particular, we shall reduce our description to ITP approaches, as PS approaches usually either do not have a unified mechanism to prove atomic formulas, or these mechanisms are in some sense -- concerning the explanatory character mentioned below -- similar to ITP approaches. The basic difference of our approach with other ITP approaches is that $\mathcal{CM}$ has a unique mechanism for formulae containing purely universally quantified variables, as well as for formulae containing existentially quantified variables. In fact, we treat formulae not containing existentially quantified variables as if they contained some existentially quantified variables (see the notion of an abstract argument in the $\mathcal{CMC}$ [20]). Moreover, the $\mathcal{CMC}$ is a procedure, and so at each step it knows what has to be done and what has to be obtained. This is important for lemmas' discovery in two ways.

First, in case of a failure, $\mathcal{CMC}$ provides explanations suitable for further analysis, and a part of missing lemmas generated in $\mathcal{CM}$ are the result of such an analysis. In other words, the second difference of $\mathcal{CM}$ to other approaches is that our basic mechanism provides suitable explanations. Here, the reader may argue that some of ITP approaches can be said to provide explanations. For instance, a critical pair in rewrite systems may be considered as an explanation. However, very often only an expert in rewrite systems is able to capture the essence of a critical pair expressing a failure, and thus the automatization of recovery process requires not only an expert in ITP, but in rewrite systems as well, while $\mathcal{CMC}$ provides explanations that allowed us, as developers, to formalize recovery process from failures due to $\mathcal{CMC}$. Moreover, this formalization does not rely on understanding problems in rewrite systems, but in ITP only.

Second, the tools used for $\mathcal{CMC}$ are finite, non-trivial procedures that may be incomplete theoretically speaking. Thus, $\mathcal{CMC}$ may fail due to incompleteness of these tools. This indicates another way of generating missing lemmas in $\mathcal{CM}$. Such lemmas are here to complete solutions. Let us note that, to this purpose, $\mathcal{CM}$ uses

also inductive tools similar to those developed in Machine Learning. A user-independence of these inductive tools is assured by a deduction-induction cycle, described in [10]. We may say that the use of non-usual inductive tools to automatize ITP makes one more difference of our approach with respect to other ones.

Thus, we have described above two ways of generating lemmas in $\mathcal{CM}$. The first one corresponds to recovering from the failures specific to $\mathcal{CMC}$. The second one corresponds to recovering from the failures specific to the tools used by $\mathcal{CMC}$. *Failure analysis* [9] takes care of failures that are assumed to be a consequence of incompleteness of our basic $\mathcal{CM}$-construction techniques. Various tools and techniques (inductive, as well as deductive) are proposed for recovery.

There are two other ways of generating lemmas.

Since $\mathcal{CM}$ treats not only atomic formulae, the third natural way to generate lemmas is to analyze the problems that may arise while proving by induction other forms of formulae, and as we have mentioned above, it is important to know to deal with implications [16].

Finally, the last way in which lemmas are generated in $\mathcal{CM}$ corresponds rather to implementing certain kind of tricks, the application of which may be suggested just on a syntactical analysis of a situation met during a proof, or which express usual attempts to simplify inductive proofs, such as finding a non-inductive proof. These heuristics may in no way be characterized as those leading to fancy and interesting proofs typical for clever people, simply because $\mathcal{CM}$ heuristics are not so particular as to rely on the semantic of particular background knowledge. This is why we say that $\mathcal{CM}$ heuristics are logically based, and this is why $\mathcal{CM}$ cannot be depicted as intelligent, since it will be very seldom that its output is the same as the output of a clever human. Since these heuristics are considered more or less as speeding up the theorem proving procedure, when an application of a heuristic from this set fails, the failure is not considered as a significant one. This means, that $\mathcal{CM}$ differs between significant and non-significant failures, similarly to ordinary work of a mathematician pictured in introduction. This explains somewhat why $\mathcal{CM}$ is able to discard a failure to prove a lemma generated in course of a proof of a given theorem, and to look for a recovery, for instance, attempting to use another tool or to reformulate the problem leading to this lemma. This illustrates also that capturing syntactically the development of an inductive proof plays a non-trivial role in $\mathcal{CM}$, and the concept of *environment* and *environment analysis* is related to this part of $\mathcal{CM}$, for brevity, non described further here.

This explains where formal creativity of $\mathcal{CM}$ comes from. In [11] we describe also the basic results we have achieved in a user-independent automatization of ITP.

Let us note that our research on $\mathcal{CM}$ is still not completed. However, section 4.5 justifies our conviction that $\mathcal{CM}$ makes a real step towards user-independent automatization of ITP, and may already now be considered as a basis of a methodology for ITP, in the sense that further research may only improve and extend our basic results. In other words, what we have done may still be considered as imperfect or not worked out enough, but it cannot be considered as useless or unsound for a methodology of ITP. Saying it in a different way, our results may already help students to provide standard proofs.

4.4 Implementation

Our system *PRECOMAS*, presented in [23], [21], is an experimental implementation of our *CM* methodology. Due to a lack of manpower, the present state of implementation reflects only part of our approach (see more in [11]).

4.5 Performance

The most important experiment of *CM* is solving the prime factorization problem presented in [22]. As we explain there, no other ITP or PS approach is able to solve this problem from the axioms presented there without a clever user, while *CM*, or an ordinary student following *CM*, succeeds from this set of axioms as well from the sets of the axioms used by other approaches. Let us mention that *CM* generates about one hundred lemmas -- some of them it fails to prove -- however, succeeding to recover from these failures. In order to show that we did not succeed simply because we are rather acquainted with natural numbers, let us mention the success of *CM* to solve rather a non-trivial planning problem as shown in [10]. Moreover, the *CM* solution is different from the solution proposed in [27] for which a human expert is claimed to be necessary. Other successful experiments and references to them are given in [10].

5 Further Work

We still work on the failure and environment analysis of *CM*. The experimental work we have performed -- with *PRECOMAS* and by hand -- validates the utility of our methodology and the benefits of software implementation. Unfortunately, we presently do not have the resources for a complete implementation, which, in our estimation, will require about 20 man/year.

Our experience in building an FC system and all the research we have done in ITP leads us to believe that there is a possibility to enlarge formal creativity of *CM* to a certain human creativity (see [22] and [10]). We confess that in order to progress more quickly a help from mathematicians, logicians and computer scientists would be welcome for this extension.

Conclusion

In this paper we introduce a non-usual, and non-trivial, division of approaches to the automatization of ITP. We explain that our approach is a representative of formally creative approaches, while other existing approaches fall into human creative approaches. We explain that formal creativity of our approach is a result of providing non-standard solutions to the problems, that in our opinion, were primordial to be solved for a user-independent automatization of ITP.

In contrast to *CM*, classified more as non-intelligent ITP aiming at standard outputs only, other existing approaches often behave as a clever student, or the proofs obtained in these approaches are in some sense interesting [4]. However, these approaches may fail when our methodology succeeds. The reasons of their failures become clear as soon as one realizes that these approaches either do not tackle problems of ITP as problems of implementing a formal technique, but rather as problems of implementing clever heuristics for proving a particular class of theorems, or they represent a formalism suitable for expressing user's own heuristics, but do not propose mechanisms making this formalism alive, or they are able to run out completely a proof when the user has been clever enough to provide a suitable

knowledge in advance, for instance, by lemmas expressed in a form suitable for the techniques of the approach. We illustrate this in [22], where we compare "the ability" of three approaches to prove the prime factorization theorem.

The paper shows also that our approach is complementary to the other ones. In other words, both formally and human creative approaches have their place in the automatization of ITP. The former approaches tend to provide a methodology for standard ITP. The latter ones tend to fulfill the goal of Artificial Intelligence by automating intelligent behavior whenever possible.

References

3. R. S. Boyer, J S. Moore: A Computational Logic; Academic Press, 1979.
4. R. S. Boyer, Y. Yu: Automated Correctness Proofs of Machine Code Programs for a Commercial Microprocessor; in: CADE 11 Proc, Springer, 1992, 416-430.
6. R. L. Constable, T.B. Knoblock, J.L. Bates: Writing Programs that construct Proofs; Journal of Automated Reasoning, vol. 1, no. 3, 1985, 285-326.
9. M. Franova, A. Galton: Failure Analysis in Constructive Matching Methodology: A Step Towards Autonomous Program Synthesizing Systems; in: R. Trappl, (ed): Cybernetics and System Research '92; 1553-1560.
10. M. Franova, Y. Kodratoff, M. Gross: Constructive matching methodology: an extension and an application to a planning problem; forthcoming, 1993.
11. an extended version of this paper; forthcoming, 1993.
12. M. Franova, Y. Kodratoff: Practical Problems in the Automatization of Inductive Theorem Proving; Rapport de Recherche No.752, L.R.I., 1992.
13. M. Franova, Y. Kodratoff: How to Clear a Block with Constructive Matching methodology; in: Proc. IJCAI'91; Morgan Kaufmann, 1991, 232-337.
14. M. Franova, Y. Kodratoff: Predicate Synthesis from Formal Specifications; in: B. Neumann, (ed.): ECAI 92 proceedings, John Wiley & Sons Ltd., 1992, 87-91.
15. M. Franova, Y. Kodratoff: Predicate Synthesis from Formal Specifications: Using Mathematical Induction for Finding the Preconditions of Theorems; to appear in proceedings of NIL'92, available as Rapport de Recherche No.781, L.R.I., 1992.
16. M. Franova, Y. Kodratoff: Simplifying Implications in Inductive Theorem Proving: Why and How?; Rapport de Recherche No.788, L.R.I., 1992.
17. M. Franova, L. Popelinsky: Constructing formal specifications of predicates; forthcoming, 1993.
18. M. Franova: Program Synthesis and Constructive proofs Obtained by Beth's tableaux; in: R. Trappl, (ed): Cybernetics and System Research 2; 1984, 715-720.
19. M. Franova: *CM*-strategy : A Methodology for Inductive Theorem Proving or Constructive Well-Generalized Proofs; in Proc. IJCAI'85, 1214-1220.
20. M. Franova: Fundamentals of a new methodology for Program Synthesis from Formal Specifications: *CM*-construction of atomic formulae; Thesis, 1988.
21. M. Franova: Precomas 0.3 User Guide; Rap. de Rech. No.524, L.R.I., 1989.
22. M. Franova: A constructive proof for prime factorization theorem: A result of putting it together in constructive matching methodology; R.R. No.780, L.R.I., 1992.
23. Franova: *PRECOMAS*: An Implementation of CM; Proc. of ISSAC'90, 16-23.
24. M. Franova: Generating induction hypotheses by Constructive Matching methodology for ITP and PS revisited; Rapport de Recherche No.647, L.R.I., 1991.
25. H. Kirchner: Preuves par Complétion; Thesis, 21 June, 1985.
26. Z. Manna, R.Waldinger: A Deductive Approach to Program Synthesis; ACM Trans. on Prog. Languages and Systems, Vol. 2., No.1, January, 1980, 90-121.
27. Z. Manna, R. Waldinger: How to Clear a Block: A Theory of Plans; Journal of Automated Reasoning 3, 1987, 343-377.
29. G. Polya: How to solve it; 1957.
30. T. Skolem: The foundations of elementary mathematic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains; in: J. van Heijenoort: From Frege to Godel; 1967, 302-334.

Representing the Knowledge used during the Requirement Engineering Activity with Generic Structures

Georges Grosz, Colette Rolland

Equipe C. ROLLAND, C.R.I., Université Paris1-Sorbonne, 17 Place de la Sorbonne, 75005 Paris, FRANCE email : {grosz, rolland}@masi.ibp.fr

Abstract : Abstracting informal requirements in terms of formal specifications of the future information system is the aim of the Requirement Engineering activity (RE). It is a cognitive activity mostly based on analogical reasoning. Skilled designers reuse the experience gain from previous design when they recognise similarities between different applications. In order to build truly intelligent systems to efficiently support RE, we believe mandatory to represent the knowledge underlying this experience. Using this representation allows to re-use part of the design process and thus, brings quality as well as productivity improvements. Our solution relies on the concept of generic structure. It is based on the following observation : many real-world phenomena, apparently different, are described in the same way. A generic structure is the expression of this common denominator, it describes behavioural as well as statical aspects. In this paper, we present the concept of generic structure and how it can be used with a simple example.

1. Introduction

The term *Requirements Engineering*(RE) [1] is used for the part of the systems development effort that involves investigating the problems and requirements of the users community, the goal being developing a formal specification of the desired Information System (IS). This specification must be validated towards the end-users' needs. The succeeding development phase, where this specification is used to design and implement a working system which is verified against the specification, may then be called *Design Engineering*. Figure 1 shows the relationship between Requirements Engineering and Design Engineering. Thus, RE aims at transforming informal IS requirements into formal - or conceptual - specifications. In other words, RE is the activity during which analysts, designers and hopefully end-users try, all together, to understand, delimit and describe in formal terms the most suitable software to help efficiently end-users and decision makers in their day-to-day missions.

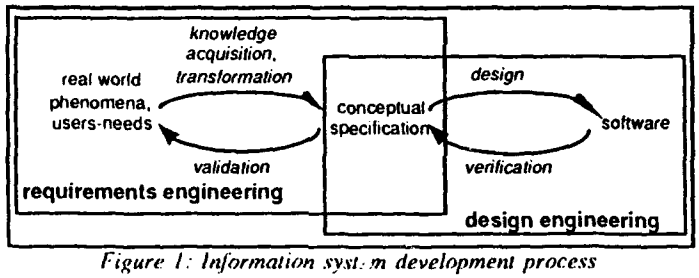


Figure 1: Information system development process

The RE activity has been recognised to be a crucial step in software development as well as in data base development [2, 3]. All these authors agreed upon the fact that when an error occurs during RE, its correction is extremely expensive and even sometimes jeopardise the entire software.
RE is a knowledge intensive task. To improve CASE capabilities, we believe it is mandatory to represent, as much as possible, the knowledge used by designers during the RE activity. Using this knowledge, designers will be able to concentrate on the perception of the real-world. The paper deals with this issue ; it tries to understand what kind of knowledge can be represented and proposes some solutions based on the concept of generic structure.
Based on such hypothesis, building a conceptual specification is about using a library of generic structures. In this new and innovative perspective, the designer does not have to redo conceptualisation efforts, he can concentrate on the perception and the understanding of the end users requirements.
The paper is structured as follows. The next section presents the concept of generic structure by showing similarities between different specifications and how a generic structure is derived. Section 3 describes examples of generic structures. In Section 4, we present the construction of a semi formal specification using generic structures This example describes the stock management in a company. Section 5 is dedicated to related works. We conclude with Section 6.

2. Generic structure : the very idea

Abstracting a formal specification from a set of requirements given by end users is mostly an analogical reasoning [4] . It is not a fully creative task. For the sake of efficiency and productivity, an experienced designer reuses previous experiences. When he recognises similar phenomena between the current requirements he is working on and previous work he did, he remembers the solution he chose and adapt it to the current problem. The knowledge used during this process is what generic structures aim at representing. The RE activity can be supported by a library of predefined generic structures adequate to some domain characteristics.
The concept of generic structure results of the two following observations. First, it exists common denominators between conceptual specifications describing similar phenomena. Second, it is possible to establish a hierarchy between classes of domain phenomena. This two observations are detail in turn.
While studying different conceptual specifications, one can realises that it exists similarities. Clearly, labels are different but common patterns can be highlighted. For instance, there is something in common between the conceptual descriptions of a car rental company, a library and a room reservation system. In a car rental company, cars are rented by customers, they bring the car back, when no car are available, the request is put in a waiting list, etc.. In a library, copies of books are checked out by subscribers, they return what they borrowed, when a book is not available, the loan request is put in a waiting list, etc.. In an hotel, rooms are reserved, when patrons leave the room, its room becomes available again, when no rooms are available, the request is put in a waiting list, etc.. In those different applications, the car, the copy of a book and the room belong to the same domain class : the "resource" class ; customers consume resources, they bring them back, when they cannot have what they want, the request is put in a waiting list. A close look to the conceptual specifications shows that they have a common underlying structure. As shown in figure 2, this is what is called generic

structure. A generic structure describes statical as well as dynamical properties associated to a class of domain phenomena.

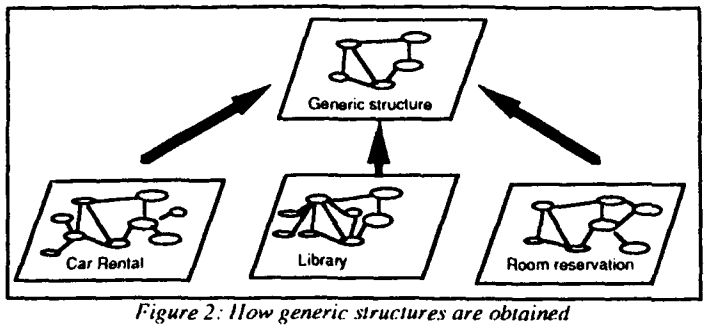


Figure 2: How generic structures are obtained

The process of identifying generic structures is a difficult abstraction process. It requires to detect similar patterns in different specifications, identify the underlying domain class, rename in generic terms the elements of the generic structure.

3. Generic structures : some examples

RE involves two kinds of knowledge: domain knowledge and model knowledge.
Domain knowledge deals with application domains. It comprises information about the application domain objects and the rules, either static or behavioural, which describe different possible states of objects. End-users are the real experts of the application domain. However, they are usually not able to build conceptual specifications since they are not familiar with model knowledge.
Model knowledge deals with conceptual formalisms. It is the knowledge about a set of concepts and the rules defining how these concepts can be used to construct a conceptual model. It is the designer's work to represent the application domain using a conceptual formalism. Model knowledge is also about being able to transform a specification expressed with one model into an equivalent one expressed with another model (e.g., translating an E/R diagram into an Object-Oriented schema).
Generic structures combine both kinds of knowledge. They are the expression of classes of domain phenomena using a model or the expression of possible transformation from a model to another one. The former type is called "Domain dependent" and the latter "Model dependent". In this paper, we focus on domain dependent generic structures. Model dependent generic structures are described in [5].

3.1. Domain dependent generic structures

In [6], many different types of domain dependent generic structures are described. We define simple basic generic structures (they deal with state changes, times changes) and also compound generic structures (they deal with resources and with the management of waiting order concerning resources). In this paper we only provide a single example of domain dependent generic structure : the generic structure describing the general behaviour of a resource. In the following, we first precise the concept of resource and informally describe its generic behaviour. We end this section with the presentation of the corresponding generic structure.

A resource is often dealt with in information system to denote available supply that can be drawn upon when needed. In other words, a resource is an entity which is required to be available for the execution of a process, and becomes unavailable when the process is using it. In a library, for instance, the entity "copy of a book" is a resource for the "loan" process while the entity book itself is not a resource because it is not involved in the loan process. In a personal computer network with a shared printer, the entity "printer" is a resource while the process is the "printing of a document".
When a *resource* is available, it can be consumed when requested by a consumer. As soon as it is consumed, it becomes unavailable for the other possible consumers because a re-source is at the exclusive use of the consumer who obtained it. When a *resource* becomes unavailable (curative management) or will become unavailable (preventive management), it may be useful, under some conditions, to *ask* for more from the supplier. For instance, when the inventory of a product becomes low or zero, an order may be automatically sent to the supplier. Re-ordering may have no meaning if the availability of a resource is set once for all at the creation time (e.g., a printer in a local network). A *resource* must be *created* (its entry in the system corresponds to a creation). Its supplier can be internal to the application (e.g., a company building products) or external (e.g., the suppliers of a super market). Complementary, a *resource* should be *deleted*. Figure 3 shows the generic structure corresponding to the behaviour of a resource.
To describe this generic structure, we use a model defined in [7]. This model is based on the notion of actor, event and entity ; they are represented by double rectangle, pentagon and circle respectively. The dashed rectangle stand either for an actor or an entity. In this model, the notion of event includes both the modifying action and the triggering conditions. Events are triggered either by actor(s) or entity(ies) ; events modify either actor(s) or entity(ies), they are the target of the pentagon. It is also possible to express static relationships between entities and/or actors by drawing a line between them.

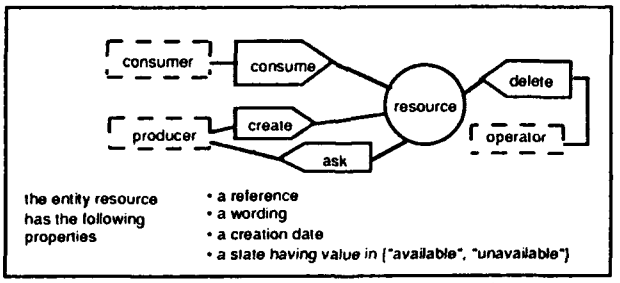


Figure 3 : generic structure for the management of a resource

The entity "resource" is modified by three events : "consume", "create" and "delete", they are respectively triggered by "consumer", "producer" and "operator". The entity "resource" triggers the event "ask" which modifies the "producer". On top of the behavioural properties related to a resource, a structural pattern is identified. A resource must have the following properties : reference, wording, creation date and a state having a value in {"available", "unavailable"}.
The study of the different types of resource management leads us to define different categories of resources. Some are consumable (for instance, the product bought by customers in a stock management system); among those, some are perishable (fresh products in a grocery) and others are non-perishable (the hardware or electronics). There are also re-usable resources (for instance the cars in a rental company). In this case, one

can distinguish between renewable (books in a library or cash in a bank) and repairable resources (cars from a rental company or planes from an airline). A resource can be characterised by more than one of the mentioned characteristics. Figure 4 shows how the different characteristics are organised in an "is_a" hierarchy. The ⊗ symbol, borrowed from the NIAM notation, expresses that the two classes are disjunctive.

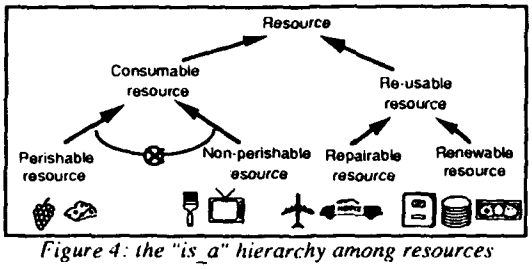


Figure 4: the "is_a" hierarchy among resources

For each classes, we defined a generic structure describing the specific behavioural and structural properties, they are detailed in [6]. The way we build them is the same as the one we used to build the general one (figure 3), (i.e., to manually define the particular behaviour and properties and express it using generic structures). Each class of the hierarchy inherits the behaviour and properties of its super classes.
For instance, the class of "perishable resource" is associated to the generic structure depicted in figure 5. It shows that (1) when a perishable resource arrives, an operator triggers an event "take delivery", this event create the resource and initialise the agenda to the sell by date, (2) the agenda triggers the event "past sell-by date" and change the state of the resource to "out dated". Thus, the statical structure of a resource comprises a "sell-by date" and the state property can have the value "out dated".

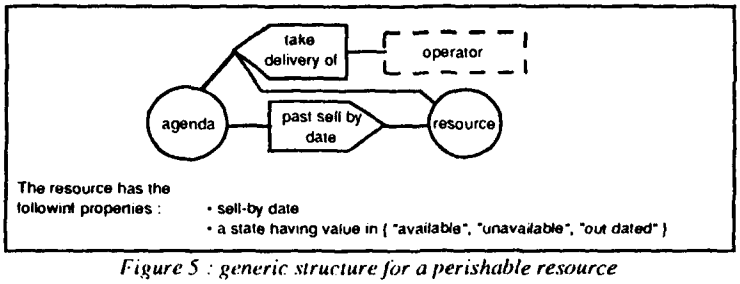


Figure 5 : generic structure for a perishable resource

4. Using generic structures

In this section, we present the general methodology for using generic structures and develop an example showing the construction of a specification describing partially the stock management of products in a company.

4.1 Methodology

The generic structures presented in the previous sections can be supported by a library of generic structures. This will change the nature of Requirement Engineering activity

itself: the designer does not have to express the real-world phenomena using a formal model but to identify those phenomena in the current application domain and instantiate the generic structures associate to those phenomena to build specifications.
The use of generic structures consists of the following steps:

A) *Recognising a phenomenon of the application domain as an instance of one of the itemised classes of phenomena and in the mean time selecting the corresponding generic structure*. Until now, the recognition of the phenomena described by domain dependent generic structures remain at the designer's charge.
B) *Defining the correspondences between the elements described in the generic structure and those which already belong to the current specifications*. While doing so, instantiation rules are highlighted, they will have to be used in a consistent way, e.g., use the same renaming rule for the same terms in the different generic structures, avoid naming conflicts (using the same name for two different purposes) and so on.
C) *Understanding the elements which have not been set in correspondence in the previous step and find, if they exist, their equivalent in the application domain*. This step leads to the improvement of the understanding of the application domain and may lead to the discovery of new issues which can lead to go back to step (A) to instantiate additional generic structures.

The process of using domain dependent generic structures ends when the designer can no longer identify itemised real world phenomena. The designer has then to complete the current specification with the specific elements of the application domain not yet described with the generic structures.
The RE activity is a decision process. The designer makes decisions all the time. In the first step, he has to recognise application domain phenomena as instances of generic structures, and this is a decision. In the second step, the decisions concern the renaming and then in the third step, decisions dealing with refinement. In [8], we demonstrate how such decisions can be handled in a knowledge based formalism (i.e. the triplet <situation, decision, action>) and successfully used in a CASE tool.

4.2 The stock management example

We consider a company selling products to its customers. Customers request are made trough orders. This first description of the application domain is shown in Figure 5.

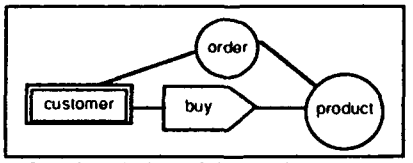


Figure 5 : A first description of the stock management company

In the following, we apply the methodology presented in the previous paragraph to the instantiation of the generic structure depicted in figure 3.

Step A : The entity "*product*" is recognised as belonging to the class "*resource*". The generic structure described in Section 3.1 has to be instantiated.

Step B : The correspondences are the following :
"*resource*" is the entity "*product*";
"*consumer*" is the actor "*customer*";
"*consume*" is the event "*buy*".

Step C : The following elements are renamed and added to the specification :
"*producer*" is identified as the actor "*supplier*";
"*produce*" and "*ask*" are identified as the events "*deliver*" and "*re-order*", respectively;
"*operator*" is identified as the actor "*warehouseman*";
"*delete*" is identified as the event "*withdraw*".

The result of the instantiation of the generic structure is shown in Figure 6. For the sake of clarity, note that we don't show the instantiation of the statical aspect described by the generic structure (figure 3)

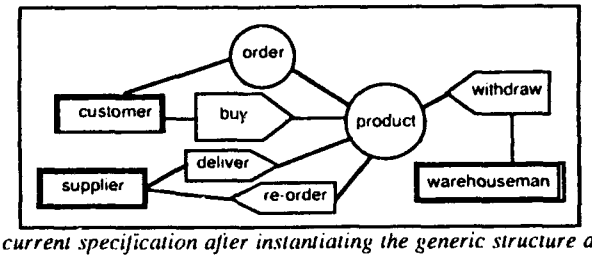


Figure 6: the current specification after instantiating the generic structure about resource

Assume the products sold by the company are perishable. This assertion leads to the following.

Step A : The entity "*product*" is recognised as belonging to the class "*perishable resource*". The generic structure described figure 5 has to be instantiated.

Step B : The correspondences are the following :
"*resource*" is the entity "*product*";
"*operator*" is the actor "*warehouseman*";

Step C : The following elements are renamed and added to the specification :
the event "*take delivery of*" is identified as the events "*reception*";
the agenda keeps its name as well as the event "*past sell-by date*"

The result of the instantiation of the generic structure is shown in Figure 7.

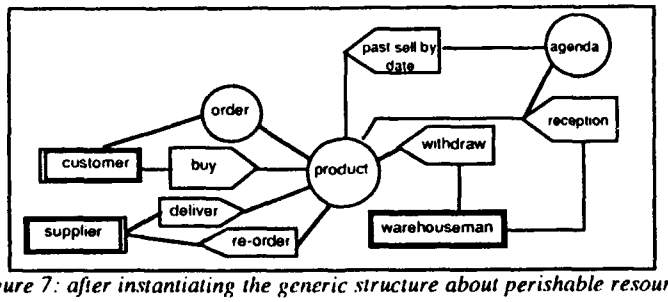


Figure 7: after instantiating the generic structure about perishable resource

Assume the designer does identify no other real world phenomena as belonging to the itemised classes. Then, the current specification must be completed according to the end-users needs (e.g., to describe the delivery system,...). To keep the example simple, we skip this step.
However, when the specification is completed towards the conceptual description of the users needs, the designer should invest some effort in the discovery of new generic structures. This is a difficult task. He has to concentrate on the elements of the specifications which has not been generated with the generic structures. A close look on those elements should able him to identify new classes of real world phenomena, express the corresponding generic structure and finally add them at the right place in the hierarchy.

5. Related Work

Some attempts to effectively guide the data base designer towards the design of a conceptual model can be found in the research literature. In the SPADE project [9], this is achieved by means of a "navigation knowledge dictionary". This dictionary helps the designer to choose between design tasks but does not help to build specifications. Within the ITHACA project, the RECAST tool [10] uses a dictionary and guides the designer to reuse object specifications. However, the designer has to modify old specifications and adapt them to the current application. Another approach, experienced in the two expert systems OICSI [11] and SECSI [12], generates part of the conceptual schema from a natural language description of the application domain. In both cases, the result depends on the completeness and accuracy of the natural language description. In [13], a learning-from-examples strategy has been developed to induce form properties and functional dependencies from day to day documents. Extensions of this work, done by Talldal [14], allow to generate an extended E/R schema. This approach does not help to specify any kind of behaviour. In [4], Maiden and Sutcliffe propose analogy as a means of reusing specifications from a CASE repository during the requirements analysis. An Intelligent Reuse Advisor, based on cognitive models of specifications reuse and automatic analogous reasoning, helps the designer to retrieve and customise existing components.
Our work goes on the same line as Reubenstien's and Waters' works (with the Requirements Apprentice [15]) where they propose clichés as a mean for building specifications with a global approach.
In [16], the authors try to make software development knowledge explicit. They propose "*to base the software development process on the reuse of descriptive models of domain-specific knowledge and prescriptive models of software engineering knowledge*". The domain models they propose are based on the same ideas than generic structures.
Similar approaches to generic structures have been developed in software engineering community. In [17], the authors propose to use a software base for enhancing the flexibility of software systems by reusing generics programs. They identify classes of *applications and programs lattices*. The instantiation of these lattices allow to generate programs. In the PARIS system [18], *partially interpreted schemas* are used to produce programs. A partially interpreted schema is a program that has some portions of it remaining abstract or undefined. Generic skeleton of programs are re-used. In [19], the concept of *design template* is used to represent classes of tasks which are characterised by their structure. Generic algorithm specification as well as generic data interface information are proposed to build programs. To some extent, applications lattices, partially interpreted schemas as well as design templates are to the production of programs what generic structures are to the production of conceptual specifications.

However, none of the above approaches have tried to explicitly represent the design knowledge used by designers as generic structures do. We believe that generic structures are well adapted to conceptual data base design. This activity is not guided by the goal (the specifications to be built) but by the source (the end-users requirements). The approach we propose does not force the designer to identify a goal but to identify the source then to instantiate generic structures for the representation of this phenomena. This kind of re-use is different from the re-use of software components or object schema as experimented, for instance, in the ITHACA project [20]. In fact, these two kinds of re-use are complementary and should be combined in a CASE tool.

6. Conclusion

In this paper, we presented a new concept for the requirement engineering activity: generic structure. A generic structure is a formal and generic expression of a class of real-world phenomena, apparently different but designed in the same way. A generic structure is independent of a particular application. A generic structure allows to re-use the process by which specifications are build and not the specifications themselves. The designer does not have to redo the conceptualisation effort for the listed phenomena, he simply instantiates the generic structures.

Two types of generic structures are mentioned in this paper ; they correspond to two types of knowledge used during the RE activity, namely, domain and model knowledge. In this paper, we focus on domain dependent knowledge and the associated generic structures. The formalism use to describe generic structures is based on actor, event and entity. However, generic structures are not model dependent, they can be used to build any kind of specification. In [21], they are expressed with the Z language.

We have prototyped the idea of generic structures using the shell of the OICSI expert system [11]. The experience has shown it to be a promising direction for research.

We want to stress that generic structures are classes of phenomena too ; and hence, it is possible and interesting to develop generic structures for generic structures. These meta-generic structures would characterise the concept of generic structure and how to create and use them. We will investigate this in future research.

The domain dependent generic structures we put forward does not cover every real-world phenomena. This study will be extended to other types of phenomenon. We are currently studying how generic structures could be discovered using a learning by example tool, with existing specifications as input.

Coupling our work with the classical approach for re-using software components is a promising research direction. It allows one to cover the whole data base development cycle. The problem is to study how generic structures can match software components to build specific projects.

References

1. E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, A. Rifaut: "A Knowledge Representation Language for Requirements Engineering", Proceedings of the IEEE, Vol. 74, Nb. 10, pp. 1431-1444, 1986.
2. B.W. Boehm: "Verifying and Validating .Software Requirements and Design Specifications", IEEE Software, Vol. 1, Nb 1, January 1984.
3. A. M. Davis : "Software requirements, analysis and specification", Prentice Hall ed, 1990.
4. N. Maiden and A. Sutcliffe: "Specification Reuse by Analogy", proceedings of 2nd Workshop on the Next Generation of CASE tools, Trondheim, Norway, May 1991.

5. G. Grosz, J. Brunet : "Transformation of Informal Entities into Object Schemas using Generic Structures", submitted paper.
6. G. Grosz: "Formalisation des connaissances réutilisables pour la conception des systèmes d'information", PhD dissertation University Paris 6, December 1991.
7. G. Grosz: "Building Information System Requirements using Generic Structures", to appear in the Proceedings of the 16th COMPSAC Conference, Chicago, September 1992.
8. G. Grosz, C. Rolland: "Knowledge Support for Information System Design", in the proceedings of "The 5th Jerusalem Conference on Information Technology", pp 261-268, October 1990, Jerusalem, Israel.
9. V. Seppanen, M. Heikkinen, R. Lintulampi, "SPADE - Towards CASE Tools That Can Guide Design", proceedings of the CAiSE 91 Conference, pp 222-239, Trondheim, Norway, May 1991.
10. M. G. Fugini, B. Pernici: "RECAST: A Tool for Reusing Requirements, in Advanced Information Systems Engineering", B. Steiner, A. Solvberg, L. Bergman (eds.), Springer Verlag Lecture Notes In Computer Science, 1990.
11. Rolland, C., Proix, C.: "An Expert System Approach to Information System Design", in IFIP World Congress 86, Dublin, 1986.
12. M. Bouzeghoub, G. Gardarin, E. Metais : "Database Design Tool: an Expert System Approach," Proc. of the 11th VLDB Conference, Stockolm, August 1985.
13. M. V. Mannino, V. P. Tseng: "Inferring Data base Requirements from Examples in Forms", 7th International Conference on Entity-Relationship Approach, pp 1-25, Rome, Italy, 1988.
14. Talldal B., Wangler B.: "Extracting a Conceptual Model from Examples of Filled in Forms", Proc. of Int. conference. COMAD, pp 327-350, N. Prakash (ed), New Delhi, India, December 1990.
15. H.B. Reubenstein, R.C. Waters: "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", IEEE Transactions on Software Engineering, Vol. 17, Nb. 3, pp 226-240, March 1991.
16. G. Arango, P. Freeman : "Modeling knowledge for software development", in Proc. 3rd Workshop on Software Specification and Design, IEEE Computer Society Press (ed), London, August 1985, pp63-66.
17. R. T. Mittermeir, M. Oppitz: "Software Bases for the Flexible Composition of Applications Systems", IEEE Transaction on Software Engineering, Vol 13, No 4, pp440-460, April 1987.
18. S. Katz, C.A. Richter, K.S. The : "PARIS : A System for Reusing Partially Interpreted Schemas", in the proceedings of the 9th Software Engineering Conference, pp 377-384, 1987.
19. M. T. Harandi, F. H. Young : "Template Based Specification and Design", in Proc. 3rd Workshop on Software Specification and Design, IEEE Computer Society Press (ed), London, August 1985, pp94-97.
20. B. Pernici, G. Vaccari, K. Villa: "INTRES: INTelligent REquirements Specification", proceedings IJCAI'89 Workshop on Automatic Software Design, Detroit, Michigan, USA, August 1989.
21. G. Grosz, C Kung, C. Rolland : "Generic Structures : A Reuse-Based Approach to Conceptual Design of Information Systems", proceedings WITS conference, Dallas, Texas, December 1992.

Development of a Programming Environment for Intelligent Robotics*

Stefano Caselli[1], Antonio Natali[2] and Francesco Zanichelli[1]

[1] Dipartimento di Ingegneria dell'Informazione
Università di Parma - Italy
[2] Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna - Italy
e-mail: mczane@verdi.eng.unipr.it

Abstract. Given the wide range and diversity of proposed architectures for autonomous robotic agents, an essential role can be played by a programming environment not hardwired to any particular architecture. This paper discusses the design of a programming environment for robotic systems, promoting rapid prototyping techniques for building different robotic architectures, while retaining a good efficiency in robot control. The environment has been conceived as the integration of a real-time robotic machine with a full-fledged logic-based system.

1 Introduction

Maybe the most critical and challenging issue in designing robotic systems working in dynamic, unstructured environments, is related to the definition of their architecture. The term architecture is used here to designate the logical organization of a system constituted by a network of (autonomous) agents that can achieve goals, under real time constraints, in not completely predictable environments, by using non-perfect sensors and actuators. Each architecture proposes a particular methodology encompassing issues such as function/competence decomposition and interaction, computational homogeneity/heterogeneity, decision responsibility distribution and static/dynamic optimization of the bounded computational resources available.

Literature reports on a wide spectrum of approaches to the design of robot architectures. Classical approaches promoted hierarchical architectures [1, 2], with the provision for different abstraction and response time levels. The main design principle is functional decomposition, together with a strict vertical control/data flow among the components, looping through perception, modeling, planning, execution and motor control. These efforts are loosely related to traditional AI in the sense that the highest level is typically a planner, which deals with a symbolic representation of robot actions and of the world. More recently,

* This work has been partially supported by CNR "Progetto Finalizzato Robotica", "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo", and Special Program N. 92.00065.CT12.

research on mobile robots has led to the development of so-called reactive and behavior-based architectures [3, 4], which are inspired by the principle that intelligence is determined by the dynamics of interaction with the physical world. Implicit integration into higher-order robot capabilities of simple task achieving behaviors, exploiting direct sensor information instead of intermediate representations, can be regarded as the major design tenet of this line of research. Since both approaches have their strengths and weaknesses, the most recent trend is to integrate reaction and planning by proposing hybrid architectures [5, 6].

Given such a wide range and diversity of architectures, an essential role can be played, at least for those who are devoid of any constraint dictated by philosophical committment, by a programming environment not hardwired to any particular architecture. The main goal of such an environment should be to enable users to exploit rapid prototyping techniques to build different kinds of robotic architectures, while retaining good efficiency in the robot control. However, designing a robotic system is not only a matter of defining its logical architecture. It also requires dealing with hardware interfacing, drive system, real-time servos, smooth trajectory interpolation and similar concrete, time-consuming issues which concur in determining the basic capabilities and dexterity upon which "intelligent" behavior is built. Under a pragmatical point of view, the environment whose development is presented in this paper is also intended to eventually become a general workbench to exercise AI techniques to a controlled-complexity robotic domain as well as to support the design and simulation of robotic workcells.

The paper is organized as follows. In section 2 the requirements, both general and specific, of a programming environment for robotics are discussed. The two main components of the environment are then presented in sections 3 and 4. Section 5 describes the overall programming model of the integrated environment whose development is currently in progress, whereas in section 6 the prototyping of a sample application is illustrated. A brief discussion summarizes the paper.

2 Requirements and Issues

Since the goal is the definition of an integrated environment to explore issues in high-level as well as in low-level robot control, besides general software engineering requirements:

- *open, extensible organization*
- *full integration among subcomponents*
- *scalability in the computational resources handled*

the environment should be characterized by the following domain-specific attributes:

- *integration of different computational and programming paradigms*
- *simulation capabilities for preliminary evaluation*
- *metaresoning facilities*

The multi-paradigm requirement seems mandatory when one aims at developing hybrid architectures, which generally partition the system into different levels exploiting explicit declarative-style knowledge along with subsymbolic task-achieving reactions.

Different design approaches have to be compared mostly in an experimental way, since general formal methods of verification are not available. For a preliminary assessment of performance, simulation is considered an almost necessary tool. Besides the visual representation of robot structure and kinematics, task execution in virtual workspaces can give insights in the robot operational behavior to an extent which can be varied according to the accuracy of the simulation. Whatever degree of accuracy is targeted, sensory information has to be generated starting from models of the physical objects (including the sensors themselves) and the interaction among them, the robot and other agents in the robot workspace.

The ultimate performance evaluation of the design comes from the interaction of the physical robot with its actual environment. Thus, an integrated environment should support a straightforward transition from simulation to real execution. This gap may be reduced when integration issues are enforced from the beginning in the design of the environment, in particular employing the same control interface and referring to the same set of commands for the simulated and the real robot. Moreover, since adaptation to changes in the world is one of the main requirements of intelligent, sensor-based robots, sensor data has to be made available to user programs, hence the set of simulated sensors should correspond as far as possible to the set of user available sensors.

Finally, the environment should provide an explicit representation of the agent architecture in order to enhance the system with some degree of self-awareness and metaresoning. Such a feature enables a uniform approach to planning, monitoring, debugging and system reconfiguration.

Current robotic programming tools are generally far from being adequate to fullfil these requirements since they are too heavily committed to low-level control and real-time issues, even though a number of recent proposals attain considerable results in restricted areas [7, 8, 9].

Programming tools should instead provide a unifying framework, allowing users to exploit a spectrum of software technologies, from efficient real-time programming to reconfigurable knowledge-based symbolic programming. The approach we are following in this research is to conceive a robot programming environment as the integration of a symbolic system constituted by a full-fledged logic environment with a flexible real-time system explicitly designed for robotic control.

As our primary application domain is dextrous manipulation and exploration using robot arms and hands [10, 11], our real-time technology is based on a control system explictly conceived for such a kind of robotic applications, namely RCCL, the Robot Control C Library [12]. RCCL offers a number of advantages over industrial controllers which will be discussed in section 3 of the work.

As regards the knowledge-based technology, our choice has been to exploit a programming environment based on an extension of Prolog with features for dynamic modularity and concurrency. Such an extension, called here *CPUX* Communicating Prolog Units with conteXts) allows users to exploit Prolog as a system programming language.

The aim of this work is to describe how such an integration is achieved and how the resulting system can be exploited to define many relevant abstractions involved in robotic architectures and extendible programming tools.

3 The Real-Time Machine

RCCL provides a robotic virtual machine for low-level control, both powerful and extendible, fully exploiting available hardware resources, mechanics and sensoriality, and yet abstracting a great deal of implementation details not relevant to high-level control and task planning or programming.

Basically, RCCL is a library of routines for describing and controlling robot positions and actions, combined with a trajectory generator for realizing these actions [12]. An RCCL application program is written in "C" and uses special primitives to specify robot action requests and queue them for servicing by the trajectory generator. Thus, RCCL is primarily a library of kinematics and control functions along with a run-time support for task execution.

RCCL has been chosen as the execution architecture for the programming environment (i.e., for implementing the low-level control law) since it provides:

- development in a flexible and friendly environment (UNIX, C, workstation) rather than in an obsolete and rigid industrial robot controller using an awkward, unflexible language;
- possibility of creating special sensor interfaces, complex tasks, synchronizations among multiple robots or other agents;
- well-defined standard with excellent documentation and a number of installation sites worldwide.

RCCL is supplied with a simple graphic simulation tool, *robotsim*. In *robotsim* the programmed task is visualized and can be evaluated with respect to kinematics and workspace features. The peculiar value of *robotsim* lies in its strict integration with the RCCL control structure. Thus there is no gap between actual or simulated execution of programmed tasks, as the very same control structure is exploited for both. Due to its flexibility, RCCL permits programming of higher complexity tasks than afforded by typical industrial controllers. By virtue of extensions we have done to the simulator, even these highly complex tasks may be directed to either the real robot or its simulation in an almost transparent manner.

Our extensions to *robotsim*, although only based on geometric modeling of objects, have allowed the simulation of simple grasp procedures as well as the generation of contact and distance sensor feedbacks which are routed to user

RCCL programs. The resulting *robotsim-II* also includes support for the presence of active objects, with customizable motion law and for interactively modifying the environment. While the overall simulation model appears limited, it seems useful for experimenting and evaluating robot architectures and strategies in a controlled-complexity domain. The integrated environment supports straightforward configuration of the simulated task, including robot sensor set and workspace.

4 The *CPUX* Programming System

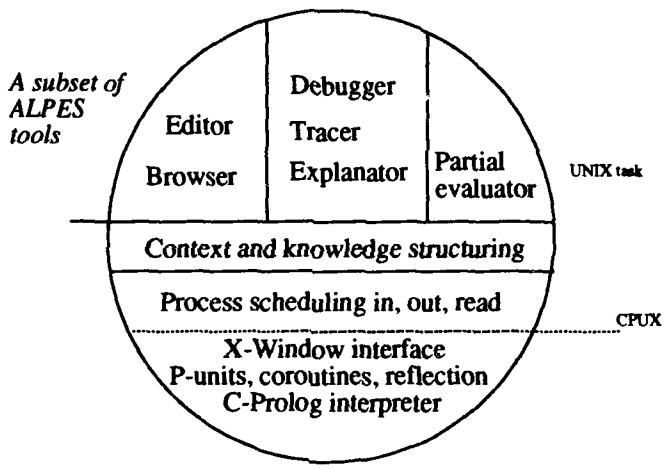


Fig. 1. The *CPUX* system and its relationship with ALPES

The environment supports (and is itself based on) an extension of Prolog with features for dynamic modularity and concurrency. Such an extension, called *CPUX* (Communicating Prolog Units with conteXts) [13], allows users to exploit Prolog as a system programming language. As concerns concurrency, agents can be explicitly created and destroyed, each agent working as an autonomous *CPUX* machine. Mechanisms for agent interaction are provided which can support mailbox-based communication models as well as generative models a là Linda [14]. Agents sharing the same processor are managed by a scheduler entirely written in *CPUX*. Process interaction policies can be expressed as meta-programs, without changing object-programs or the supporting machine. As concerns modularity, *CPUX* fully supports contextual logic programming [16], a programming paradigm in which software components can be statically and dynamically configured by composing modules (*CPUX* units) into structures, called contexts. Through contexts, users can prototype new abstractions and exploit these prototypes to create other prototypes in an incremental way, very similarly to object-oriented programming.

The most significant architectural features of *CPUX* and the related programming methodologies have been exploited for prototyping an integrated, open and evolving programming environment, in the ALPES Esprit project 973 of the European Community [17]. The relationship between *CPUX* and ALPES is illustrated in Fig. 1.

5 The Environment Programming Model

The simple integration of ALPES and RCCL around a client-server interaction scheme has proven extremely valuable even limiting to a "conventional" robot programming perspective, i.e. when the purpose of the environment is to assist users in developing and testing robot applications in the framework of robot- or task-level programming [18]. Efficiency concerns are addressed mostly in the RCCL control subsystem, although recent work has reported on performance levels of compiled Prolog comparable to those obtained by leading C compilers [19].

The integration between the two subsystems has provided immediate results without sacrificing the features originally offered by RCCL. Some of these results are related to the *CPUX* system language, while others are related to ALPES features and tools. *CPUX* features are particularly suited to prototyping and also to debugging at a high level of abstraction, a very crucial activity in developing robotic applications. ALPES tools, written in *CPUX* as modular, extendible components, have been specialized to build high-level interfaces. In particular, the ALPES X-Window browser has been extended to create a tool which allows the user to define interactively the initial setup for task execution under the *robotsim-II* simulator, based on database modules containing taxonomies of known robots, available sensors and machines/parts to be chosen and included in the task.

However, the main result of the integrated environment is to promote a programming model founded on the concept of an agent society in which agents can interact through a shared abstraction called *world* (fig. 2). This multiagent model emphasizes autonomy and heterogeneity of agents in a Distributed Artificial Intelligence perspective, by amalgamating different granularities and architectures. Agents can be heterogeneous: the only requirement is that any agent has to agree on the abstract communication interface constituted by the *world* (implemented as a *CPUX* communication unit). To this purpose, the RCCL real-time machine has been extended in order to define a special agent, the RCCL agent, which, acting as a server, looks for a command in the *world* and executes it.

In conclusion, this system makes general-purpose abstractions and mechanisms regarding important aspects of robot programming immediately available, such as agent interaction, agent configuration and design, and meta-control of agent behavior.

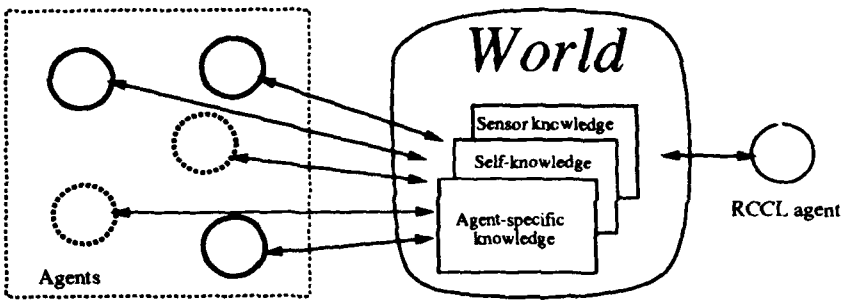


Fig. 2. Overall framework of the system

5.1 Agent Interaction

Information stored in the *world* can represent sensor-related knowledge, self-knowledge and agent-specific knowledge.

Sensor knowledge is constituted by high-level representation of sensor data. In our current environment it is automatically updated by the low-level extended RCCL system at user-defined rates. Self-knowledge represents knowledge on the network configuration and on the state of agents, e.g. computational state (waiting for a message, waiting for a processor), execution time, period *et cetera* (this information is used by the *CPUX* scheduler). Agent-specific knowledge is constituted by messages explicitly produced and consumed by agents in order to cooperate or interact. Starting from the world knowledge, each agent can create local world models, to be used in a shared or private way. Thus, this architecture is particularly suited to the design and development of hybrid robot architectures. However, it does not exclude the possibility of experimenting behavioral approaches, since the notion of *world* generalizes a concept of world only comprising external physical sensory input.

CPUX process interaction mechanisms support agent interaction models similar to the generative communication of Linda, and are expressed by the following primitive predicates:

- `world -? msg(M)`
  like *in*, withdraws the message from the *world* with possible suspension
- `world -?? msg(M)`
  like *rd*, reads the message from the *world* ;
- `world -! msg(m)`
  like *out*, adds the message to the *world* .

These mechanisms can express both cooperative and competitive agent interaction schemes. However, more specific mechanisms and concepts can be required in robot applications and the environment accordingly extended.

For example, in full competitive interaction schemes, the global control-law has to emerge from a collection of different, independent, local laws. In order to achieve such a behavior, some (meta) control level or decisor component can

be introduced. Once the designer has chosen how to implement the decision system, a new abstraction should be made available in the system. Such a new communication abstraction can be implemented in *CPUX* as shown in [15]. In fact, CPU processes are abstractions of independent coroutines which can be explicitly transfer control each other and all *CPUX* process interaction operators are themselves written in Prolog.

5.2 Agent Configuration and Design

Robot applications demand for a wide spectrum of agent types ranging from purely deductive goal-directed activities, to completely behavior-based ones.

Since *CPUX* support both declarative and procedural programming styles, such a spectrum of agents classes can be easily expressed. Contextual programming provides a powerful tool to dynamically compose agents as a hierarchy of different knowledge units (open or closed to variations). In fact, through the extension operator `U >> G`, goal G is solved in a new context obtained by adding U clauses on top of current context. In this way, an agent like a classical STRIPS-like planning system can be dynamically configured by creating a context composed of a unit implementing the algorithm core with a unit containing domain specific status and operators descriptions.

In several domains some agents have to be defined as periodic activities possibly with fixed deadline. Such a class of agents can be implemented in *CPUX* thanks to the possibility of handling asinchronous events (timing interrupts). Moreover, suitable environment tools can be defined to help programmers during the design phase to check for the existence of a feasible schedule. At this level, the typical techniques of constraint logic programming can be exploited.

In order to increase modularity and flexibility, a class of dynamically configurable agents can be required in order to merge goal-oriented decision procedures with the opportunistic decisions related to the current state of the world. Again, contextual mechanisms can constitute the basic support. For example:

```
agent-B :-
  world -?? situation(Current),
     ; Get global knowledge for the recognized situation
  B-default >> Current >> g.
agent-B:-
  B-default >> g.
```

Agent with goal **agent-B** evolves in dependence of a specific recognized situation, if any, generated by another "perceptual" agent. In performing activity **g**, it takes into account the generic **B-default** assumptions only if more specific and relevant ones are not available.

5.3 Meta Control

Another critical part of multiagent robot systems is agent network configuration, which can be estabilished both in a static way, as a consequence of user design

or as a planning activity, and in a dynamic way, as a consequence of new user commands or of a meta-control layer. Meta-level programming features of *CPUX* (todemo/odemo reflection mechanisms described in [21]) can be exploited to allow users to modify current behavior at two different levels:

1. at agent level without modifying the overall architecture (network of agents);
2. at decision network level, without modifying agent local behavior.

Assume for example that **agent-1** is currently in charge of issuing commands to the RCCL agent. Later, it has to compete with the decisions of a new **agent-2**. This kind of dynamic reconfiguration can be done without modifying **agent-1**, by dynamically associating to it a meta-unit **meta-1** in which a meta-rule performs the routing of **to-rccl** message to a decisor component. In general, meta-rules have the following form:

```
todemo(CurrUnit,AuxReg,CurrProcess, CurrGoal) :- <body>.
```

which means "to demonstrate" the **CurrGoal** within **CurrProcess** perform the **<body>** actions. These actions can express different kinds of meta-control policies, such as:

- route the message to another agent;
- send another message;
- kill (another) process;
- monitor the agent behavior;
- ...

6 Prototyping an Application

In this section we introduce a simple example of robotic application and schematically show how the integrated environment and its mechanisms can assist the user in the development process. In particular, only a few relevant aspects of the overall task are presented.

The application consists of a task for a robot manipulator, combining the exploration of an unknown workspace and the manipulation of blocks. By means of its sonars, the robot has to recognize a number of block stacks and to obtain, via blocks manipulation, a new stacks organization. The environment is thus very similar to a classical blocks world, even though important extensions that can be handled include presence of obstacles and dynamically changing environment.

Figure 3 shows a typical initial situation for the simulated manipulator task. The robot in this example is configured to have two sonar range sensors, one aligned with the gripper (z-sonar), the other rotated by 90 degrees (x-sonar).

The problem requires safe hand navigation in an unknown workspace, while perceptual activities attempt to determine the configuration of blocks in the workspace. For sake of simplicity some conditions are assumed along with some *a-priori* knowledge: the workspace is monodimensional and obstacles are higher than any block stack, since they are actually containers of blocks.

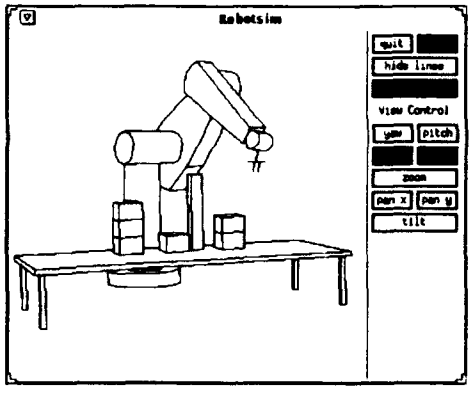


Fig. 3. An example of initial workspace

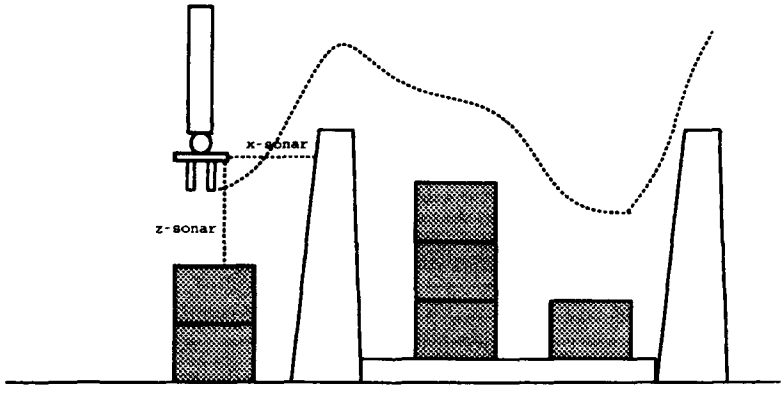


Fig. 4. Safe navigation over the blocks

Agent **follow-height** drives the robot hand safely through the workspace, adjusting the direction of the trajectory so as to mantain the hand over the blocks. Agent **detect-obstacle** monitors the front sonar to detect the presence of obstacles ahead. When some obstacle is perceived, detect-obstacle sends a message to the *world* informing other agents that a container is either approaching or getting far. The perceptual interpretation of the workspace, which in this case aims at obtaining a world representation in terms of block stacks, is a good example of contextual agents. In fact, the **perceive-blocks** agent derives the configuration of blocks from z-sonar data differently in context **no-container** than it does in context **over-container**, on account of the presence of the bottom.

Finally, as soon as perception has taken place, planning activity can be carried out while reactive navigation can be stopped. Agent **strips** is triggered with a context **blocks** to synthesize the sequence of robot actions which can reach the user-defined goal state. The list of operators is then sent to the *world*,

where an **execute** agent transforms this high-level description into movement messages for the RCCL agent.

7 Conclusions

The paper has presented the organization of an environment for prototyping and experimenting advanced robotic architectures. The environment is built upon the integration of a real-time robotic machine with a full-fledged logic system. Among the distinguishing features of the environment are:

- possibility of simulation of sensor-driven tasks, exercising a real-time controller which is identical to that of the physical robot,
- general mechanisms for agent interaction which may implement various architectural paradigms currently investigated for intelligent robots,
- efficiency in rapid prototyping and in exploration of architectural alternatives, due to the underlying logic-based environment enriched with concurrency, modularity, and contexts.

In this work we focused our attention on mechanisms rather than on well founded robot programming models, since defining such models is precisely the goal of several research efforts in robotics. Contextual mechanisms and low-level process control primitives have been exploited to provide a flexible and extendible basic architecture, supporting both robot-level and task-level programs built according to prototyping techniques. Since the same mechanisms can be used to build the robot programming environment, the application layer can grow up in a synergetic and uniform way with its own user-support.

While these mechanisms proved suitable for structuring purposes, a still open issue in this approach is the degree of real-time the symbolic level of the system is to be affected.

References

[1] Albus J., McCain H., Lumia R.: NASA/NBS Standard Reference Model Telerobot Control System Architecture, NIST Tech. Note 1235, NIST, Gaithersburg, MD, July 1987.

[2] Crowley J. L.: Coordination of Action and Perception in a Surveillance Robot, IEEE Expert, Vol. 2(4), Winter 1987.

[3] Brooks R. A.: A Robust Layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, Vol. 7, Vol. RA-2, No. 1, 1986.

[4] Mataric M. J.: Integration of Representation Into Goal-Driven Behavior-Based Robots, IEEE Trans. on Robotics and Automation, Vol.8, No. 9, September 1992.

[5] Gatt E.: Integrating Reaction and Planning in a Heterogeneous Asynchronous Architecture for Mobile Robot Navigation, Proc. of AAAI Symposium on Integrated Intelligent Architectures, ACM Sigart Bulletin, Vol. 2. No. 4, August 1991.

[6] Lyons D. M., Hendriks A.J.: Planning for Reactive Robot Behavior, IEEE Int. Conf. on Robotics and Automation, Nice, France, May 1992.

[7] Miller D. J., Lennox R. C.: An Object-Oriented Environment for Robot System Architectures, IEEE Int. Conf. on Robotics and Automation, Cincinnati, OH, May 1990.

[8] Fagg A.H., Lewis, M.A., Iberall,T., Bekey G.A.: R^2AD: Rapid Robotics Application Development Environment, Int. Conf. on Robotics and Automation, Sacramento, CA, April 1991.

[9] Chen C.X., Trivedi M.M., Bidlack C.R.: Simulation and Graphical Interface for Programming and Visualization of Sensor-Based Robot Operation, Int. Conf. on Robotics and Automation, Nice, France, May 1992.

[10] C. Bonivento, E. Faldella, G. Vassura, "The University of Bologna Robotic Hand Project: Current State and Future Developments", ICAR'91, International Conference on Advanced Robotics, Pisa, Italy, June 1991.

[11] Caselli S., Faldella E., Zanichelli F.: Grasp Synthesis for a Dextrous Robotic Hand Using a Multi-Layer Perceptron, IEEE Melecon '91, Ljubljana, YU, May 1991.

[12] Hayward V., Paul R. P.: Robot Manipulator Control under Unix - RCCL: A Robot Control "C" Library, Int. Journal of Robotics Research, Vol. 5, No. 4, 1986.

[13] Mello P., Natali A.: Programs as Collections of Communicating Prolog Units, ESOP-86, LNCS n. 219, Springer Verlag 1986.

[14] Gelernter D.: Generative Communication in Linda, ACM Trans. on Programming Languages and Systems, Vol. 7, No. 1, January 1985.

[15] Mello P., Natali A.: Extending Prolog with Modularity, Concurrency, and Metarules, New Generation Computing, Vol. 10, No. 4, August 1992.

[16] Monteiro L., Porto A.: Contextual Logic Programming, Proc. 6th ICLP, Lisbon, Portugal, The MIT Press, 1989.

[17] ALPES Esprit Project n. P973. Final Technical Report, September 1989.

[18] Lozano-Pérez T.: Robot Programming, Proc. of the IEEE, Vol. 71, No. 7, 1983.

[19] Roy P. V, Despain A. M.: High Performance Logic Programming with the Aquarius Prolog Compiler, Computer, Vol. 25, No 1, January 1992.

[20] Brooks R. A., "The Behavior Language: User's Guide", AI Lab Memo No. 1227, April 1990.

[21] Cavalieri M., Lamma E., Mello P., Natali A.: Meta-Programming through direct introspection: a comparison with meta-interpretation techniques, in "Meta-Programming in Logic Programming", The MIT Press, Cambridge, MA, Abramson and Rogers eds, 1989.

On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification*

Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "la Sapienza"
Via Salaria 113, 00198 Roma, Italia
e-mail: aschaerf@assi.ing.uniroma1.it

Abstract. Most of the work regarding complexity results for concept languages consider subsumption as the prototypical inference. However, when concept languages are used for building knowledge bases including assertions on individuals, the basic deductive service of the knowledge base is the so-called instance checking, which is the problem of checking if an individual is an instance of a given concept. We consider a particular concept language, called $\mathcal{ALE}$ and we address the question of whether instance checking can be really solved by means of subsumption algorithms in this language. Therefore, we indirectly ask whether considering subsumption as the prototypical inference is justified. Our analysis, carried out considering two different measure of complexity, shows that in $\mathcal{ALE}$ instance checking is strictly harder than subsumption. This result singles out a new source of complexity in concept languages, which does not show up when checking subsumption between concepts.

1 Introduction

Concept description languages (also called terminological languages or *concept languages*) have been introduced with the aim of providing a simple and well-established first order semantics to capture the meaning of the most popular features of the structured representations of knowledge. In concept languages, concepts are used to represent classes as sets of individuals, and roles are binary relations used to specify their properties or attributes.

It is a common opinion that subsumption checking (i.e. checking whether a concept represent necessarily a subset of the other) is the central reasoning task in concept languages. This has motivated a large body of research on the problem of subsumption checking in different concept languages (e.g. [2, 4, 13]). However, if concept languages are to be used for building knowledge bases including assertions on individuals, the basic deductive service of the knowledge base is the so-called

*This work has been supported by the ESPRIT Basic Research Action N.3012-COMPULOG and by the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council).

instance checking, which is the problem of checking whether a set of assertions implies that a given individual is an instance of a given concept.

In this paper we address the question of whether instance checking can be easily solved by means of subsumption algorithms and whether considering subsumption as the prototypical inference is justified.

The outcome of the analysis is that the answer to the question in the general case is *no* and that there are cases where instance checking is strictly harder than subsumption. In fact, the instance checking problem for the considered language ($\mathcal{ALE}$) turns out to be of higher complexity than the subsumption problem. This result is all the more interesting since $\mathcal{ALE}$ is *not* some artificial language (just defined for thew purpose of showing that the complexity of subsumption may differ from the one of instance checking) but a rather natural language whhich has been investigated before.

This result singles out a new source of complexity in concept languages, which does not show up when checking subsumption between concepts. A practical implication of this fact is that any actual deduction procedure for reasoning on structured knowledge representation systems cannot be based solely on subsumption checking, but has to embed some reasoning mechanisms that are not easily reducible to subsumption. This fact must be considered when designing the deductive services of a system for the development of applications based on concept languages.

In our analysis we have assumed we are dealing with complete reasoners.On the contrary, most of the existing systems (e.g. LOOM[9], CLASSIC[11], BACK[12]), use incomplete reasoners[1] (except for $\mathcal{KRIS}$[1]) and in these systems a careful analysis of the relationship between subsumption and instance checking is lacking. Anyway, in our opinion (supported also by other researchers, e.g. see [13]), the study of complete methods gives more insight on the structure of the problem and it is useful also for the development of better incomplete reasoners.

Furthermore, we measure the complexity of instance checking by the measures suggested in [14]. *Data complexity* (i.e. the complexity w.r.t. the size of the knowledge base) and *combined complexity* (i.e. w.r.t. both the size of the knowledge base and the size of the concept representing the query) [2]. Another result of our analysis is that there are cases where the complexity obtained using the two measures differs substantially. This result proves that the distinction is necessary and must be taken into account in order to understand the behavior of the system in practical cases.

The paper is organized as follows. In Section 2 we provide some preliminaries on concept languages, in particular on the language $\mathcal{ALE}$. In Section 3, we deal with the problem of instance checking in $\mathcal{ALE}$. Discussion and conclusions are drawn in Section 4. For the sake of brevity, some proofs are only sketched.

2 The Language $\mathcal{ALE}$

We consider the language $\mathcal{ALE}$ [3, 13], which is an extension of the basic concept language $\mathcal{FL}^-$ introduced in [2]. Besides the constructors of $\mathcal{FL}^-$, $\mathcal{ALE}$ includes

[1] CLASSIC is actually complete, but with respect of a non standard semantics

[2] In [14] it is also considered the so-called *expression complexity* (i.e. the complexity w.r.t. the size of the concept representing the query), but it seems less interesting in our setting.

qualified existential quantification on roles and complementation of primitive concepts.

Given an alphabet of primitive concept symbols $\mathcal{A}$ and an alphabet of role symbols $\mathcal{R}$, *$\mathcal{ALE}$-concepts* (denoted by the letters C and D) are built up by means of the following syntax rule (where R denotes a *role*, that in $\mathcal{ALE}$ is always primitive)

$$
\begin{array}{lll}
C, D \longrightarrow & A \mid & \text{(primitive concept)} \\
& \top \mid & \text{(top)} \\
& \bot \mid & \text{(bottom)} \\
& \neg A \mid & \text{(primitive complement)} \\
& C \sqcap D \mid & \text{(intersection)} \\
& \forall R.C \mid & \text{(universal quantification)} \\
& \exists R.C & \text{(qualified existential quantification)}
\end{array}
$$

We have chosen the language $\mathcal{ALE}$ for several reasons. On one hand, it is rich enough to express a significant class of assertions, as we will see in the examples. In addition, it is suitable to express concepts representing meaningful queries. In particular, due to the full existential quantification it is possible to express queries requiring some form of navigation in the knowledge base through the assertions on roles. For example, the query: "find the individuals who have a friend whose child is a doctor" can be expressed as $\exists$`FRIEND`.$\exists$`CHILD`.`Doctor`. On the other hand, $\mathcal{ALE}$ avoids other expressive constructors (such as disjunction of concepts) which introduce other sources of complexity and whose treatment is out of the scope of this paper.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following equations are satisfied:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow\ d_2 \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge\ d_2 \in C^{\mathcal{I}}\}
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a *model* for a concept C if $C^{\mathcal{I}}$ is nonempty. A concept is *satisfiable* if it has a model and *unsatisfiable* otherwise.

We say C is *subsumed* by D, written as $C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$; and C is *equivalent* to D, written as $C \equiv D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every $\mathcal{I}$.

The construction of knowledge bases using concept languages is realized by permitting concept and role expressions to be used in assertions on individuals. Let $\mathcal{O}$ be an alphabet of symbols denoting individuals, an *$\mathcal{ALE}$-membership assertion* (or simply assertion) is a statement of one of the forms:

$$C(a) \quad \text{or} \quad R(a, b)$$

where C is an $\mathcal{ALE}$-concept, R is a role, and a, b are individuals in $\mathcal{O}$.

In order to assign a meaning to the assertions, the interpretation function $\cdot^{\mathcal{I}}$ is extended to individuals in such a way that $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual $a \in \mathcal{O}$ and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (Unique Name Assumption). The meaning of the above assertions is now straightforward: if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation, $C(a)$ is satisfied by $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $R(a, b)$ is satisfied by $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A set Σ of $\mathcal{ALE}$-assertions is called an *$\mathcal{ALE}$-knowledge base*. An interpretation $\mathcal{I}$ is said to be a *model* of Σ if every assertion of Σ is satisfied by $\mathcal{I}$. Σ is said to be *satisfiable* if it admits a model. We say that Σ *logically implies* an assertion α (written $\Sigma \models \alpha$) if α is satisfied by every model of Σ.

In the so-called terminological systems, the knowledge base also includes an intensional part, called *terminology*, expressed in terms of concept definitions. However, almost all implemented systems assume that such definitions are acyclic, i.e. in the definition of concept C no reference (direct or indirect) to C itself may occur. It is well known that any reasoning process over knowledge bases comprising an acyclic terminology can be reduced to a reasoning process over a knowledge base with an empty terminology, (see [10] for a discussion on this reduction and its complexity). For this reason, in our analysis we do not take into account terminologies and, therefore, we conceive a knowledge base as just a set of $\mathcal{ALE}$-assertions.

Definition 2.1 *We call* instance checking in $\mathcal{ALE}$ *the following problem: given an $\mathcal{ALE}$-knowledge base Σ, an $\mathcal{ALE}$-concept D, and an individual a in $\mathcal{O}$, check if $\Sigma \models D(a)$ holds. Furthermore, denoting with $|\Sigma|$ the size of Σ and with $|D|$ the size of D, we call* data complexity *(resp.* combined complexity*)* of instance checking in $\mathcal{ALE}$ *the complexity of checking if $\Sigma \models D(a)$ w.r.t. $|\Sigma|$ (resp. $|\Sigma|$ and $|D|$).*

Notice that instance checking is a basic tool for more complex services. For example, using instance checking it is possible to solve the so-called *retrieval* problem: given a concept D, find all the individuals that are instances of D. Retrieval can be performed simply by iterating the instance checking problem for all the individuals in Σ.

Example 2.2 Let Σ_1 be the following $\mathcal{ALE}$-knowledge base:

```
{CHILD(john,mary), (∀CHILD.¬Graduate)(john),
(Female⊓∃CHILD.Graduate)(mary)}
```

It is easy to verify that Σ_1 is satisfiable. Notice also that the addition of the assertion `Graduate(mary)` to Σ_1 would make the knowledge base unsatisfiable; in fact, due to the first two assertions, $\Sigma_1 \models \neg$`Graduate(mary)`. □

3 Instance checking in $\mathcal{ALE}$

In [3] it is shown that checking the subsumption relation between two $\mathcal{ALE}$-concepts is an NP-complete problem and that checking the satisfiability of an $\mathcal{ALE}$-concept is coNP-complete. With regard to instance checking, we know that it is at least as hard as subsumption. In fact, subsumption can be reduced to

instance checking in the following way: given two concepts C and D, the subsumption test $C \sqsubseteq D$ is performed by checking whether the knowledge base composed by the single assertion $C(a)$ (where a is any individual) logically implies $D(a)$, i.e. checking if $\{C(a)\} \models D(a)$ holds. This result holds for instance checking w.r.t. the combined complexity. In fact, the complexity of subsumption is measured w.r.t. the size of both the candidate subsumee and the candidate subsumer, which becomes respectively part of the knowledge base and part of the query.

With regard to the data complexity, we know that a concept C is unsatisfiable if and only if the knowledge base $\{C(a)\}$ implies every assertion, i.e. if $\{C(a)\} \models B(a)$ (where B is any concept). It follows that concept unsatisfiability can be reduced to instance checking w.r.t. data complexity (the size of B can be obviously fixed). Since concept satisfiability is coNP-complete, concept unsatisfiability is NP-complete. We can therefore conclude that instance checking is NP-hard even w.r.t. the data complexity[3].

We now give a more precise characterization of the complexity of instance checking, w.r.t. both the data complexity and the combined complexity.

We start with a lower bound for the data complexity of instance checking in $\mathcal{ALE}$. As shown above, instance checking in $\mathcal{ALE}$ is NP-hard. We prove that it is coNP-hard too. Since subsumption in $\mathcal{ALE}$ is NP-complete, a consequence of this result is that (assuming NP $\neq$ coNP) instance checking for $\mathcal{ALE}$ is strictly harder than subsumption.

This unexpected result shows that the instance checking problem in $\mathcal{ALE}$ suffers from a new source of complexity, which does not show up when checking subsumption between $\mathcal{ALE}$-concepts. This new source of complexity is related to the use of qualified existential quantification in the concept representing the query that makes the behavior of the individuals heavily dependent on the other individuals in the knowledge base. The following example enlights this point.

Example 3.1 Let Σ_2 be the following $\mathcal{ALE}$-knowledge base:

$\{$FRIEND(john,susan), FRIEND(john,peter),
LOVES(susan,peter), LOVES(peter,mary),
Graduate(susan), $\neg$Graduate(mary)$\}$.

Consider now the following assertion

β = $\exists$FRIEND.(Graduate$\sqcap\exists$LOVES.$\neg$Graduate)(john).

Asking whether $\Sigma_2 \models \beta$ means asking whether John has a graduate friend who loves a not graduate person. At the first glance, since Susan and Peter are the only known friends of John, it seems that the answer the query is to be found by checking whether either $\Sigma_2 \models$ Graduate$\sqcap\exists$LOVES.$\neg$Graduate(susan)
or $\Sigma_2 \models$ Graduate$\sqcap\exists$LOVES.$\neg$Graduate(peter) is true.

Since $\Sigma_2 \not\models$Graduate(peter)
it follows that $\Sigma_2 \not\models$ Graduate$\sqcap\exists$LOVES.$\neg$Graduate(peter),

[3]Notice that this result implies the previous one. In fact, the combined complexity is obviously higher or equal to the data complexity.

and since $\Sigma_2 \not\models \exists$**LOVES.**$\neg$**Graduate(susan)**
it follows that $\Sigma_2 \not\models$ **Graduate**$\sqcap\exists$**LOVES.**$\neg$**Graduate(susan)**.

Reasoning in this way would lead to the answer NO. On the contrary, the correct answer to the query is YES, and in order to find it, one needs to reason by *case analysis*. In fact, the query asks if in every model M of Σ_2 there is an individual, say **a**, such that **FRIEND(john,a)**, **Graduate(a)** and $\exists$**LOVES.**$\neg$**Graduate(a)** are true in M. Obviously, in every model M of Σ_2, either **Graduate(peter)** or $\neg$**Graduate(peter)** is true. In the first case, it is easy to see that **a** is simply **peter** (and the not graduate person he loves is **mary**), while in the second case **a** is **susan** (and the not graduate person she loves is just **peter**). Therefore, such an individual **a** exists in every model of Σ_2, and the query gets the answer YES.

Therefore, even if none of the individuals related to the individual **john** through the role **FRIEND** is in the condition requested by the query, it happens that the combination of the assertions on the individuals (**susan** and **peter**) in the knowledge base is such that in every model one or the other is in that condition. □

The previous example shows that, in order to answer to a query involving qualified existential quantification, a sort of *case analysis* is required. We now show that this kind of reasoning makes instance checking in $\mathcal{ALE}$ coNP-hard w.r.t. the data complexity.

The proof is based on a reduction from a suitable variation of the propositional satisfiability problem (SAT) to instance checking in $\mathcal{ALE}$. We define 2+2-CNF formula on the alphabet P, a CNF formula F such that each clause of F has exactly four literals: two positive and two negative ones, where the propositional letters are elements of $P \cup \{true, false\}$. Furthermore, we call 2+2-SAT the problem of checking whether a 2+2-CNF formula is satisfiable. Using a reduction from 3-SAT (here omitted) we derive the following result: 2+2-SAT is NP-complete.

Given a 2+2–CNF formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$, we associate with it an $\mathcal{ALE}$-knowledge base Σ_F and a concept Q as follows. Σ_F has one individual l for each letter L in F, one individual c_i for each clause C_i, one individual f for the whole formula F, plus two individuals *true* and *false* for the corresponding propositional constants. The roles of Σ_F are Cl, P_1, P_2, N_1, N_2, and the only primitive concept is A.

$$
\begin{aligned}
\Sigma_F \;=\; \{ \;& Cl(f,c_1), Cl(f,c_2), \ldots, Cl(f,c_n), \\
& P_1(c_1, l^1_{1+}), P_2(c_1, l^1_{2+}), N_1(c_1, l^1_{1-}), N_2(c_1, l^1_{2-}), \\
& \ldots \\
& P_1(c_n, l^n_{1+}), P_2(c_n, l^n_{2+}), N_1(c_n, l^n_{1-}), N_2(c_n, l^n_{2-}), \\
& A(true), \neg A(false) \; \} \\
Q \;=\; & \exists Cl.(\exists P_1.\neg A \sqcap \exists P_2.\neg A \sqcap \exists N_1.A \sqcap \exists N_2.A).
\end{aligned}
$$

Intuitively, checking if $\Sigma_F \models Q(f)$ corresponds to checking if in every truth assignment for F there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e. a clause that is not satisfied.

Lemma 3.2 *A 2+2-CNF formula F is unsatisfiable if and only if $\Sigma_F \models Q(f)$.*

"$\Rightarrow$" Suppose F is unsatisfiable. Consider any model $\mathcal{I}$ of Σ_F, and let $\delta_{\mathcal{I}}$ be the truth assignment for F such that $\delta_{\mathcal{I}}(l) = true$ if and only if $l^{\mathcal{I}} \notin A^{\mathcal{I}}$, for every letter l. Since F is unsatisfiable, there exists a clause C_i that is not satisfied by $\delta_{\mathcal{I}}$. It follows that all the individuals related to c_i through the roles P_1, P_2 are in the extension of $(\neg A)^{\mathcal{I}}$ and all the individuals related to c_i through the roles N_1, N_2 are in the extension of $A^{\mathcal{I}}$. Thus $c_i^{\mathcal{I}} \in (\exists P_1.\neg A \sqcap \exists P_2.\neg A \sqcap \exists N_1.A \sqcap \exists N_2.A)^{\mathcal{I}}$, and consequently $f^{\mathcal{I}} \in Q^{\mathcal{I}}$. Therefore, since this argument holds for every model $\mathcal{I}$ of Σ_F, we can conclude that $\Sigma_F \models Q(f)$.
"$\Leftarrow$" Suppose F is satisfiable, and let δ a truth assignment satisfying F. Let $\mathcal{I}_\delta$ be the interpretation for Σ_F defined as follows:

- $A^{\mathcal{I}_\delta} = \{l^{\mathcal{I}_\delta} \mid \delta(l) = true\}$,
- $\rho^{\mathcal{I}_\delta} = \{(a^{\mathcal{I}_\delta}, b^{\mathcal{I}_\delta}) \mid \rho(a,b) \in \Sigma_F\}$ for $\rho = Cl, P_1, P_2, N_1, N_2$.

It is easy to see that $\mathcal{I}_\delta$ is a model of Σ_F and $f^{\mathcal{I}_\delta} \notin Q^{\mathcal{I}_\delta}$. Therefore, we can conclude that $\Sigma_F \not\models Q(f)$. □

Theorem 3.3 *Instance checking in $\mathcal{ALE}$ is coNP-hard in the size of the knowledge base.*

Proof. The thesis follows from Lemma 3.2 and from the fact that, given a 2+2-CNF formula F, Q is fixed independently of F and Σ_F can be computed in polynomial time w.r.t. the size of F. □

The above reduction shows that the coNP-hardness arises even if the knowledge base is expressed using a simple language, i.e. in order to obtain intractability it is sufficient to enrich only the query language with the qualified existential quantification, keeping a tractable assertional language. This result is in contrast with the result reported in [7]: in that paper, a polynomial instance checking algorithm is provided for an $\mathcal{AL}$-knowledge base ($\mathcal{AL}$ extends $\mathcal{FL}^-$ with primitive complements) using the query language $\mathcal{QL}$ (which is an extension of $\mathcal{ALE}$). Unfortunately, as pointed out in [8], while that algorithm is sound, it is in fact not complete. In particular, it answers NO to the query β of Example 3.1.

Since the language $\mathcal{ALE}$ involves primitive complements and the above reduction makes use of them it may seem that the coNP-hardness arises from the interaction of qualified existential quantification with the primitive complements. On the contrary, we are able to show that the coNP-hardness is caused by the qualified existential quantification alone. In fact, if we consider the language $\mathcal{FLE}^-$ ($\mathcal{FL}^-$ + qualified existential quantification), we are able to prove that instance checking in $\mathcal{FLE}^-$ is coNP-hard too. The intuition is that in $\mathcal{FLE}^-$ it is possible to require a reasoning by case analysis too. In particular, this is done by considering two $\mathcal{FLE}^-$-concepts of the form $\exists R.\top$ and $\forall R.C$ (instead of A and $\neg A$ in $\mathcal{ALE}$), and exploiting the fact that their interpretation covers the entire domain, i.e. $\exists R.\top \sqcup \forall R.C \equiv \top$.

In the following we give an upper bound to the data complexity of instance checking in $\mathcal{ALE}$, proving that it is in Π_2^p.[4]

[4] The class Π_2^p, also denoted by $coNP^{NP}$, consists of the problems whose complement can be solved by a nondeterministic polynomial time algorithm exploiting a nondeterministic polynomial oracle. For a discussion on the classes Σ_k^p, Π_k^p, and Δ_k^p see for example [6].

Lemma 3.4 *Let Σ be a satisfiable $\mathcal{ALE}$-knowledge base, a be an individual, and D be a $\mathcal{ALE}$-concept. Then checking if $\Sigma \not\models D(a)$ can be done by a nondeterministic algorithm, polynomial in $|\Sigma|$, which exploits a nondeterministic polynomial oracle.*

Proof. (sketch) We know that $\Sigma \models D(a)$ if and only if the knowledge base $\Sigma \cup \{\neg D(a)\}$ is unsatisfiable. In general, $\neg D$ is not an $\mathcal{ALE}$-concept, since it contains the complement of a non-primitive concept. Therefore, $\Sigma \cup \{\neg D(a)\}$ is not an $\mathcal{ALE}$-knowledge base, but a knowledge base of a more expressive language called $\mathcal{ALC}$ (see [13]) which extend $\mathcal{ALE}$ with general complements. In [1] an algorithm is presented for checking the satisfiability of an $\mathcal{ALC}$-knowledge base. That algorithm consists of a NPTIME procedure which exploits a PSPACE subprocedure. It is possible to show that, in such algorithm, if the size of the concept which uses general complement is fixed, then the subprocedure runs in NPTIME. Since we measure the data complexity, the size of $\neg D$ can be considered fixed and the claim follows. □

Theorem 3.5 *Instance checking in $\mathcal{ALE}$ is in Π_2^p w.r.t. the data complexity*

Proof. Easily follows from Lemma 3.4 □

Notice that the complexity of instance checking w.r.t. the data complexity is not completely placed in the complexity hierarchy. We conjecture that it is Π_2^p-complete.

We now show that instance checking in $\mathcal{ALE}$ is PSPACE-complete w.r.t. combined complexity. The following lemma states the PSPACE-hardness of the problem and its long proof can be found in [5].

Lemma 3.6 *Instance checking in $\mathcal{ALE}$ is PSPACE-hard w.r.t. the combined complexity.*

Theorem 3.7 *Instance checking in $\mathcal{ALE}$ is PSPACE-complete w.r.t. the combined complexity.*

Proof. Lemma 3.6 states the PSPACE-hardness Since instance checking in $\mathcal{ALE}$ is obviously easier than the PSPACE-complete problem of instance checking in $\mathcal{ALC}$ (see [1]), it follows that instance checking in $\mathcal{ALE}$ is also in PSPACE. Hence instance checking in $\mathcal{ALE}$ is PSPACE-complete. □

4 Discussion and Conclusions

The following table summarizes the complexity of instance checking in $\mathcal{ALE}$ with respect to both knowledge base complexity and combined complexity, together with the previous known result regarding subsumption.

subsumption	instance checking data complexity	instance checking combined complexity
NP-complete [3]	NP-hard coNP-hard in Π_2^p	PSPACE-complete

These results single out several interesting properties:

1. Instance checking is *not* polynomially reducible to subsumption, in the general case. As a consequence, an algorithm for instance checking based on subsumption (and classification), should include some other complex reasoning mechanisms.

2. The data complexity and the combined complexity of instance checking in $\mathcal{ALE}$ are in different classes: one in Π_2^p and the other in PSPACE-complete. This fact highlights that, in order to have an actual complexity measure of the performance of our systems, we must pay attention to which is the crucial data of the application.

3. Reasoning in $\mathcal{ALE}$ suffers from an additional source of complexity which does not show up when checking subsumption, even w.r.t. the data complexity. This new source of complexity is related to the use of qualified existential quantification in the concept representing the query which requires some sort on reasoning by case analysis. In particular, due to this source of complexity the instance checking problem (unlike subsumption) cannot be solved by means of one single level of nondeterministic choice.

4. With respect to the combined complexity, $\mathcal{ALE}$ is in the same class as more complex languages (e.g. $\mathcal{ALC}$, see [1]). Therefore, whenever the expressiveness of $\mathcal{ALE}$ is required, other constructors can be added without any increase of the computational complexity.

With regard to point 2, one may object that if the reasoning process underlying this source of complexity is not in the intuition of the user, as pointed out in example 3.1, it must be ruled out by the reasoner (and by the semantics). On the contrary there are cases in which this kind of reasoning seems to agree with the intuition and therefore it must be taken into account.

Acknowledgements

I would like to thank Francesco Donini, Maurizio Lenzerini, and Daniele Nardi for discussion that contributed to the paper and Franz Baader, Enrico Franconi, and Marco Schaerf for many helpful comments on earlier drafts. I also acknowledge Yoav Shoham for his ospitality at the Computer Science Department of the Stanford University, where part of this research has been developed.

References

[1] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1991.

[2] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence AAAI-84*, 1984.

[3] F. M. Donini, B. Hollunder, M. Lenzerini, A. Marchetti Spaccamela, D. Nardi, and W. Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327, 1992.

[4] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162. Morgan Kaufmann, 1991.

[5] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. From subsumption to instance checking. Technical Report 15.92, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1992.

[6] M. Garey and D. Johnson. *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[7] M. Lenzerini and A. Schaerf. Concept languages as query languages. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence AAAI-91*, 1991.

[8] M. Lenzerini and A. Schaerf. Querying concept-based knowledge bases. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1991.

[9] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, June 1991.

[10] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[11] P. F. Patel-Schneider, D. McGuiness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representatin system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3):108–113, June 1991.

[12] C. Peltason. The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119, June 1991.

[13] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[14] M. Vardi. The complexity of relational query languages. In *14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.

Mutual Knowledge*

S. Ambroszkiewicz,**

Institute of Computer Science, Polish Academy of Sciences, 01-237 Warsaw, ul. Ordona 21, POLAND, E-mail: sambrosz@plearn.bitnet

Abstract. The notions of agent's behavior and fundamental knowledge for decision making are introduced. Then it is shown how mutual knowledge, i.e. knowledge of the form: *agent* i_1 *knows that agent* i_2 *knows that ... tht agent* i_n *knows that ...* is used by the rational agents. This *use* is, following Wittgenstein (1958), regarded as meaning of this knowledge.
...

1 Introduction.

Let us suppose that there are two agents who are involved in a game. Each of them has its own knowledge about the game, his opponent and generally about his own world that may influence his decision. The situation is extremely interesting when these two worlds overlap. Then the agents themselves as well as their knowledge, knowledge about the knowledge, and so on, become the subjects of the knowledge. To illustrate the problem let us present a story[3] told by Jerzy Loś to Alfred Tarski as early as 1959.

> Two warships, say A and B, were fighting each other. Each of the captains of the ships (for short, captain A and captain B) hired a spy on shore, and paid him \$1 for any information of any importance to the fight. By coincidence, not so rare in reality, it happened that the two spies were one and the same person. It also happened that the radar of ship A was out of order. The spy learned it and immediately sent to captain B this news and of course got \$1. Soon afterwards he sent to the captain A the following message: *captain B knows that your radar is out of order.* Later on he sent to captain B: *captain A knows that you know that his radar is out of order.* And so on. The problem is how much money could the spy collect?

In fact the problem the story states is to what stage of iteration the mutual knowledge delivered by the spy is important to the captains. Could the spy get an infinite amount of money? The imporance means here that the knowledge influences captain's behavior.

Hence, the problem is illposed until the war game and patterns of behavior of the captains are specified. So that we must first answer the following questions: *What are possible patterns of behavior?* and *How does mutual knowledge influence these behaviors?*

* This work was supported by KBN grant No.210979101

** My thanks are due to Professor J.Łoś for inspiration and many helpful discussions.

[3] Authorized by J. Loś.

Another important problem that is raised by the story is meaning of the knowledge. Of course we can build model-theoretic or possible-worlds semantics for mutual knowledge, however the story suggests much more straighforward semantics based on the following claim:
One's knowledge is meaningful (for him) if it may influence his decision.
According to this claim the question: *What does it mean for us?* is equivalent to *How do we use it?* So that *meaningless* will be the same as *useless.*

Actually this approach to semantics is not novel. It is a version of Wittgenstein's view:

> 43. For a *large* class of cases – though not for all – in which we employ the word "meaning" it can be defined thus: the meaning of a word is its use in the language. And the *meaning* of a name is sometimes explained by pointing to its *bearer.*
>
> Wittgenstein (1958), pp.20-21

Since meaning is use and the use is to get some goal, there must be also a criterion for reaching this goal. So that we are very close the theory of decision making. In fact this Wittgenstein's *use* is nothing but playing so called *language game.*

Let us present an example of simple language game adopted from Wittgenstein (1958). Assume that there are two agents: agent 1(speaker) and agent 2(hearer). Agent 1 has the actions $\mathbf{a}_1^i \in A_1$, whereas agent 2 the actions $\mathbf{a}_2^i \in A_2$.

The actions of agent 1 are commands for agent 1 to take an appropriate action. Agent 2's knowledge has the following form: *agent 1 tells me* $\mathbf{a}_1^i$. He understand a command if in response he takes an appropriate action. Behavior of agent 2 may be described as follows:
if agent 1 tells me $\mathbf{a}_1^i$ *then I will take one action from the set* $A_2' \subseteq A_2$. Hence it is behavior that constitues the meaning of that knowledge. Let us note that this resembles somewhat the procedural representation of knowledge. Actually procedural representation defines partial behavior of the agent.

The exact meaning of commands is coordinated in learning process that consists in repetition of the game. It is worth to notice that mutual knowledge is important for this process. We may again cite Wittgenstein:

> 210. "But do you really explain to the other person what you yourself understand? Don't you get him to *guess* the essential thing? You give him examples, –but he has to guess their drift, to guess your intention."–Every explanation which I can give myself I can give to him too.–"He guesses what I am intend" would mean : various interpretations of my explanation come to his mind, and he lights on one of them. So in this case he should ask; and I should answer him.
>
> Wittgenstein (1958), pp.83-84

We may also assume that each agent has preferences on the results of the actions, however we want here to focus mainly on behavior.

Generally agent i's behavior is defined as action reduction, say B_i, such that given knowledge, say K, and set A_i of possible actions of the agent i, $B_i(A_i, K)$ is a subset of A_i.

We will assume that this behavior is defined only for *fundametal* knowledge, whereas other kinds of *relevant* knowledge (i.e. the knowledge that may influence the agent decision) should be transformable into the fundamental one. However in order to define explicite fundamental knowledge and behavior we should specify the situation like in the Loś' story. In the case of decision making these two notions are relatively straighforward defined (see the next section).

The introduction of the notions of fundamental knowledge and behavior distinguish this work from the vast literature on this subject: Hintikka 1962, McCarthy 1979, Moore 1977 and 1980, Konogile 1981, Fagin 1984, Kraus and Lehman 1986, Aiello, Nardi and Schaerf 1988, Halpern and Moses 1990 to mention only few.

Rational behavior (in the sense of decision theory) is recently appreciated as complementary to logical inferrence in reasoning, see Doyle, Levesque and Moore 1988, and Doyle 1990. In the next sexion we will specify patterns of rational behavior and war game to give a solution to the problem posed in the Loś' story.

As to semantical aspects of knowledge there is growing need among AI community for a notion of meaning independent of those delivered by Tarskian and *possible-worlds semantics*, *see for example* Winograd and Flores 1986, Rosenfield 1988, Clancey 1991, Hewitt 1991 and Gasser 1991. It seems that the approach to semantics based on Wittgenstein's idea is promising to fit out this need. In the last section of the paper we will try to build such semantics just by showing how an agent uses his mutual knowledge.

Actually similar idea was invented in so called *Social Conceptions of Knowledge and Actions* Gasser (1991) and *Open Information Systems Semantics* Hewitt (1991), let us quote from Gasser (1991):

> The notion that the meaning of a message is the response it generates in the system that receives it was introduced by Mead (1934) and was later used independently in the context of computing by Hewitt (1977). Using this conceptualization, a message that provokes no response has no meaning, and each message with impact has a *specific* meaning, played out as a set of specific response behaviors.

There the notion of mutual knowledge is crucial to define one of their fundamental notions, namely *webs of commitments* or *systems commitments*; let us again quote from Gasser (1991):

> This approach rests on a somewhat untraditional idea of what an agent is: in Gerson formulation, an agent is a reflexive collection of processes involved in many situations. To varying degrees, the agent – that is, some component process of the agent – can take the viewpoint of any participant in those situations.

Hence, the notions of agent's behavior and fundamental knowledge, introduced in this paper, that serve to show how the agent uses his mutual knowledge, seem to be important.

2 Patterns of rational behavior.

General form of mutual knowledge is the following:

$$\textit{agent } i_1 \textit{ knows that ... agent } i_k \textit{ that } E_k \textit{ takes place,} \qquad (1)$$

where E_k is some event and $k = 1, 2, ..., n$. This is called agent i's knowledge of depth n.

Generally, by agent's behavior we mean transformation that given a knowledge, it reduces the agent's actions. Any knowledge that may cause this reduction will be called *relevant*, so that we allow E_k in (1) to be any event that may be called relevant, even such peculiar one as spots on the Sun, if it is only shown how this influences the decision of the agent.

Let us consider the simplest yet not trivial decision problem: two-person non-cooperative game in normal form: $G = (A_1,\ A_2,\ u_1,\ u_2)$, where A_i is the set of all possible actions of agent i, and $u_i : A_1 \times A_2 \to R$ is agent i's payoff. The game is played as follows: the agents take actions a_1, a_2 recpectively, then they receive respectively payoffs $u_1(a_1, a_2)$ and $u_2(a_1, a_2)$. It is supposed that each of them wants to maximize his payoff.

In our opinion in game context there are only two kinds of relevant knowledge which are fundamental. The first one is about the payoff, whereas the second one is about the opponent's actions. The rest of relevant knowledge, whatever it may be, must be transformable onto these two kinds. The motivaton behind this is that the agent's behavior to be defined below depends only on these two kinds of knowledge, i.e. what agent takes into account for his final decision is his payoff and possible actions of the opponent because his payoff depends on action taken by the opponent.

For simplicity we assume here that each agent knows precisely his payoff.

Now let us consider agent's knowledge about the opponent's actions. Suppose that the agent 1 knows that in the play his opponent, (i.e. agent 2) will choose an action from the set $A_2^* \subseteq A_2$. Let this supposition be called *deterministic* (contrary to probabilistic one to be considered later). Then agent 1's behavior consists in reducing his set of actions to the optimum ones, say $A_1' \subseteq A_1$, given this supposition. So that we may write $A_1' = b_1(A_1,\ u_1,\ A_2^*,)$, where b_1 is to be the behavior of agent 1. This reduction b_1 (called agent 1's behavior) depends on his own payoff (it is supposed that the agent tries to maximize his payoff) and the opponents actions. Hence elements of the set A_1' are meant as the optimum player 1's actions given this supposition.

Another approach to agent's behavior is based on *probabilistic* suppositions. Assume that agent 1 has a belief on which action will be chosen by the opponent. In other words he has a priori probability distribution, say μ on the set A_2. So that for any $a_2 \in A_2$: $\mu(a_2)$ is the measure of his belief that agent 2 will choose the action a_2. Then the behavior of agent 1 consists in reducing, on the basis of this probabilistic supposition and his utility, his action set to some subset $A_1' \subseteq A_1$. So that we may write $A_1' = B_1(A_1,\ u_1, \mu)$, where B_1 is to be the behavior of agent 1. Elements of A_1' are meant as the optimum actions given this

supposition. If they maximize the expected payoff relatively to this distribution, then this behavior is called Bayesian. However the belief of agent 1 has nothing to do with the real play, that is, the game is played only once and the agent 2's choice in this play is independent on what that belief is. Since after the play agents will get pure payoffs not the expected ones, it does not seem reasonable to optimize agent 1's utility relatively to μ, that is, to play one of the actions that maximize the expected payoff. This optimization is meaningful only in the case where the game is repeated many times, but this changes the situation completely.

Despite the criticism presented above, it seems that beliefs play important role in decision making. So far as deterministic supposition may be interpreted as direct information on the opponent's choice, probabilistic supposition may be interpreted as beliefs coming from previous observations how often particular actions have been chosen by the opponent in similar situations. Once we assume that agent has his own belief before the play, it is not decided in advance that the agent must maximize the expected payoff relatively to the belief.

Although those two kinds of behavior are different, they do not contradict each other, so we may also consider the third kind being a mixture of them.

In order to touch ground, we will specify some behaviors. As for the second kind, we choose the Bayesian behavior, because so far it is the only well established behavior based on the probabilistic suppositions. Let μ be a probability distribution on A_j. For the simplicity we assume that the letters: i, j denote different agents, i.e. $i = 1$ iff $j = 2$, and $i = 2$ iff $j = 1$. Formally this behavior is defined as follows:

$$B_i^*(A_i, u_i, \mu) \stackrel{df}{=} \{a_i \in A_i : \ a_i \text{ is best response to } \mu\},$$

where a_i is best response to μ if a_i maximizes the function:

$$f(a_i) = \sum_{x \in A_j} u_i(a_i, x)\mu(x).$$

The behavior, we are going to introduce, which is based on deterministic suppositions, is the following:

$$b_i^*(A_i, u_i, A_j') \stackrel{df}{=} \{a_i \in A_i : \ \exists a_j \in A_i' \ \ a_i \text{ is best response to } a_j \text{ in } A_i \ \},$$

where a_i is best response to a_j if

$$\forall a_i' \in A_i : \ u_i(a_i, a_j) \geq u_i(a_i', a_j).$$

In words, it consists in rejecting (as not optimum) the actions that are not best responses to any action from the supposition. This behavior can hardly be considered as rational unless we assume that the agent has no risk aversion. In order to see so let us present the following agent 1's payoff table:

u_1	a_2^1	a_2^2
a_1^1	0	3
a_1^2	3	0
a_1^3	$2+\epsilon$	$2+\epsilon$

where a_1^k (resp. a_2^k) are the actions of agent 1 (resp. agent 2), and $0 \leq \epsilon < 1$. Note that if ϵ is close to 1 then the agent should choose a_1^3 that gives him $2+\epsilon$ for sure, whereas other actions might give him a bit more, that is, 3 however he can not be sure of that. The action a_1^3 is not a best response *neither to* a_2^1 *nor to* a_2^2. So that if the number 3 in the table means \$3 (a small amount of money) then despite of the risk of geting nothing, it is *reasonable* to choose a best response, i.e. a_1^1 or a_1^2. Hence the extreme behavior that consists in elimination of actions that are not best responses may be considered as the behavior of *rational agent* who is a gambler, i.e. independently from the payoff he has no risk aversion. It is still an open problem how to measure risk aversion and how this aversion influences the agent's behavior.

Now let us come back to the Loś' story. Assume that if the radar is out of order then the game G given by the following table is played:

G	b_1	b_2	b_3
a_1	$-2,3$	$-1,0$	$1,0$
a_2	$0,0$	$1,2$	$0,0$
a_3	$1,1$	$0,2$	$0,0$

where a_1, a_2, a_3 are the actions of captain A, whereas b_1, b_2, b_3 are captain B's actions. First (resp. second) number in the table is payoff to the first (resp. second) agent.

Let us assume that behaviors of the captains are b_1^* and b_2^* respectively, that is, captain A takes place of agent 1, and captain B takes place of agent 2. Assume that this fact was *common knowledge* among them in the sense of Aumann (1976). Let us note that the mutual knowledge delivered by the spy was about the game they played. In other words the event E_k in (1) is denoted as: *the game G is played.*

Although captain A did know, of course, that his own radar was out of order, i.e. the game G was played, he could not eliminate any of his actions until he got to know more about the enemy's knowledge and his behavior. When captain B got to know that the game G was played, according to his rational behavior, he removed action b_3 that is not best response in game G. When captain A got to know so, i.e. that captain B knew that the game G was played, knowing the behavior of his opponent captain A also knew how he had reduced his actions. Hence, on the basis of this reasoning, captain A removed action a_1 that is not best response if only action b_3 is removed. When, in turn, captain B got to know that a_1 was removed, he removed also action b_1, so that the only action left was b_2. When captain A got to know that his enemy took action b_2, he, of course, would take action a_2. Hence, we see that the knowledge delivered by the spy was important to the fight up to the first stage where all remaining actions were best responses. So that the spy could collect only \$4.

This shows how relevant (mutual) knowledge is transformed into fundamental knowledge; in this case it is transformation into the knowledge about opponent actions.

3 A semantics for mutual knowledge

The aim of this section is first to present another general multi-agent decision problem which confirms that the notions of agent's behavior and fundamental knowledge are essential, i.e. without these notions we can not show how agents *use* their mutual knowledge. Second, to stress again that this *use* may be regarded as meaning of this knowledge.

Suppose that there are two agents involved in a competitive game, say $G = (A_1, A_2, u_1, u_2)$. Each of the agents has a computer and a software that assists him in decision making. In other words agent i has his own computer system, say S_i. Let us consider for example S_1. It works as follows, given information on agent 1's payoff and what actions (say, from subset $A_2^1 \subseteq A_2$) may be chosen by the opponent (i.e. agent 2), system S_1 suggests agent 1 to restrict his actions to subset $A_1^1 \subset A_1$. Actually this system reflects the agent 1's behavior, i.e. it is an input-output system, such that given fundamental knowledge as an input, it generates an output being a subset of A_1.

We assume that each agent has put all his rules of behavior into his system so that he relies on it completely. For this reason we will identify agent's system with his behavior.

For simplicity we assume that each agent knows precisely his own payoff and this knowledge is stored in his system, so that all relevant knowledge should be transformable on the knowledge about the opponents actions.

Now we are ready to present another story:

- It happened that the agent 2 got a piracy copy of S_1, so that runing it on his computer he got to know the subset $A_1^1 \subseteq A_1$, i.e. the action reduction done by his opponent. Then he ran his own S_2 with the knowledge on A_1^1 as the input, reducing in this way his actions to the set A_2^2.
- Suppose that the agent 1 got to know so. However this knowledge was irrelevant to him until he has known how the opponent had used the copy of his system, i.e. until he has known the set A_2^2 which, in turn, he could compute only if he got a copy of S_2. So suppose that he managed to get a piracy copy of S_2.
- Then he was able to simulate the reasoning of his opponent, getting the set A_2^2. Next he applied his own system S_1 to A_2^2 reducing his actions to the set A_1^3.
- Suppose that agent 2 got to know that his opponent (agent 1) had a copy of his system and that agent 1 knew that agent 2 had a copy of S_1. Then he simulated the reasoning of of agent 1, so that he learned the set A_1^3. Later on with this set as input he ran his system (i.e. S_2) reducing in this way his actions to the set A_2^4.
- And so on.

Let us analize the story in detail. In this story the mutual knowledge is about agents' systems, i.e. about theirs behaviors and at the same time about the game played. In Loś story the mutual knowledge was only about the game, so that in order to get a solution we had to assume that the specified behaviors of the captains were *intuitive* common knowledge. Each of the items above represents some agent's mutual knowledge, and shows how the agent uses it. Let us put down it explicite. This knowledge may be listed as follows:

(1) *(a) agent* 2 *knows the system* S_1.
(2) *(a) agent* 1 *knows that agent* 2 *knows system* S_1,
(b) agent 1 *knows system* S_2.
(3) *(a) agent* 2 *knows that agent* 1 *knows that agent* 2 *knows system* S_1,
(b) agent 2 *knows that agent* 1 *knows system* S_2.
And so on.

It is extremely interesting here that each of the mutual knowledge alone (i.e. either *(a)* about system S_1 or *(b)* about system S_2) is important for the agent only at depth 1, but together they may be useful also at depth greater than 1.

It is worth to notice that after getting a piracy copy of S_1, the agent 2's system is no longer S_2 but a new system, say S_2^1, defined as follows:

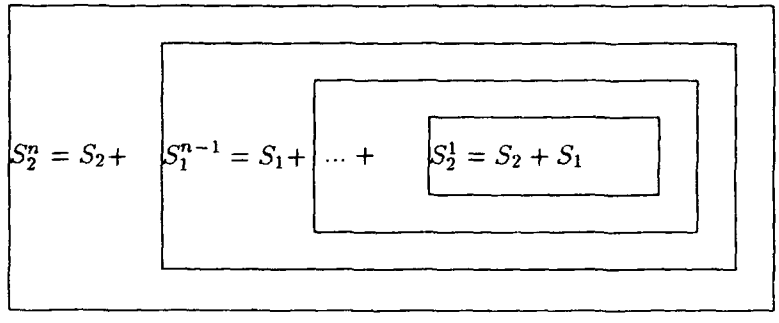
$$S_2^1 \;=\; S_2 \;+\; \text{a copy of } S_1$$

The knowledge about S_1 is transformed, just by runing system S_1, into the knowledge about agent 1's actions, which, in turn, is used as the input for S_2 to restrict agent 2's actions.

So that agent 1, after getting the copy of S_2 and knowledge that his opponent has already a copy of his system, he in fact knows the new system of agent 1, i.e. the system S_2^1. Runing it on his computer, he simulates the reasoning of agent 2, so that he gets to know the action reduction done by agent 2. This knowledge is, in turn, an input to S_1. Therefore now a new system of the agent, say S_1^1, consists of the original one (i.e. S_1) plus S_2^1.

Hence, after each step of knowledge iteration, actually, a new system is created that consists of the original one enriched by the current system of the opponent.

In this way (see the picture below) we get a nested sequence of systems resulting in use of the mutual knowledge *(a)* of depth n and *(b)* of depth $n-1$.



These systems transform mutual knowledge into the fundamental one. We claim that these nested systems represent the meaning of the mutual knowledge.

4 Conclusion

The notions of behavior and fundamental knowledge are crucial for the semantics of multi-agents systems proposed in the paper. It would be interesting to find

out general definitions for these notions.

For the sake of simplicity we have not present complete formal description of the nested systems.

The cases considered above are very special, that is, they are in fact special examples of decision problems where agents know precisely the system of the opponent. It is not clear how to build general framework even for decision problems with incomplete mutual knowledge as well as for multi-agent systems with cooperation and communication.

It seems the Wittgenstain's idea is far from being explored. Especially it is interesting how to define such semantics for complex notions.

Finally we regard mutual knowledge to be extremely important in learning and generally in multi-agent systems.

References

1. L. Aiello, D. Nardi and M. Schaefer: Yet another solution to the three wisemen puzzle. In: Proc. of ISMIS'88, Z. Raś and L. Saitta (Eds.) (1988) 398-407
2. R. J. Aumann: Agreeing to Disagree. Annals of Statistics **4** (1976) 1236-1239
3. W.J. Clancey: Book Reviewof ref.5. Artificial Intelligence **50** (1991) 241-284
4. J. Doyle, H.J. Levesque and R.C. Moore: Panel: Logicality vs. Rationality. In: Proc. of the Second Conf. on Theoretical Aspects of Reasoning about Knowledge. Morgan Kaufmann (1988) 343-364
5. J. Doyle: Rationality and its Roles in Reasoning. In: Proc. AAAI-90 (1990) 1093-1100
6. R. Fagin, J. Halpern and M.Y. Vardi: A model-theoretic analysis of knowledge: preliminary report. In: Proc. IEEE Symposium on Foundations of Computer Science. (1984) 268-278
7. L. Gasser: Social conception of knowledge and action: DAI foundations and open system semantics. Artificial Intelligence **47** (1991) 107-138
8. E. M. Gerson: Scientific work and social worlds. Knowledge **4** (1977) 357-377
9. J.Y. Halpern and Y.O. Moses: Knowledge and Common Knowledge in a Distributed Enviroment. J. ACM **37**(3) (1990) 549-587
10. C. Hewitt: Viewing control structures as patterns of passing messages. Artif. Intell. **8** (1977) 323-364
11. C. Hewitt: Open Information Systems Semantics for Distributed Artificial Intelligence. Artificial Intelligence **47** (1991) 79-106
12. J. Hintikka: Knowledge and Belief. Cornell University Press, Ithaca, New York, (1962)
13. K. Konolige: A first-order formalization of Knowledge and action for a multiagent planning system. Machine Intelligence **10** (1981)
14. S. Kraus and D. Lehmann: Knowledge, Belief and Time. In: Proc. 13th ICALP (L. Kott, ed.) Rennes, Springer, *LNCS* **226** (1986) 186-195
15. J. McCarthy: First-order theories of individual concepts of propositions. Machine Intelligence **9** (1979) 120-147
16. G. H. Mead: Mind, Self and Society. University of Chcago Press, IL, (1934)
17. R. Moore: Reasoning about Knowledge and Action. In: Proc. of IJCAI-83 (1983) 382-384

18. R. Moore: Reasoning about Knowledge and Action. SRI Int. Technical Note 337, Menlo Park, CA (1984)
19. I. Rosenfield: The Ivention of Memory: A New View of the Brain. Basic Books, New York (1988)
20. T. Winograd and F. Flores: Understanding Computers and Cognition. Albex, Norwood, NY (1986)
21. L. Wittgenstein. Philosophical Investigations. Basil Blackwell, Oxford (1958)

Expressive Extensions to Inheritance Networks*

Krishnaprasad Thirunarayan

Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435.
Email: tkprasad@cs.wright.edu

Abstract. Even though much work has gone into explicating the semantics of inheritance networks, no consensus seems to have emerged among the researchers about their precise semantics. In fact, there are several different possible interpretations of the same network topology. So, from a practical knowledge representation standpoint, in the absence of any independently verifiable semantics, we wish to pursue the approach of enhancing the language of traditional inheritance networks to enable the user to choose among the various available options, to program in a more complete description of the input problem in the enriched language. This approach permits representation of certain networks that were not representable previously, and allows making subtle distinctions among networks that were not hitherto possible. In this paper we propose an annotated inheritance network language, develop its formal semantics by amalgamating harmoniously a family of related "local" inheritance theories, and discuss some implementation issues.

1 Introduction

Inheritance networks (abbreviated as INs in the sequel) form an important class of data structures for knowledge representation. Much work has gone into explicating their semantics as evidenced by the contemporary literature. (See [1], [3], [5], [6], [7], [8], [12], [13], [15], [17], [19], [21].) However, no consensus seems to have emerged among the researchers about the precise semantics of INs. That being the case, several families of related inheritance theories have been proposed, each justified on the basis of intuitive examples. In other words, INs have been given several different semantics based on different possible interpretations of the same topology. A closer scrutiny of these issues reveals that the differences among some of these interpretations stem from the various choices that have been made about the available options [16], [22]. So, from a practical knowledge representation standpoint, in the absence of an independently verifiable semantics [12], it makes sense to enhance the language of INs to make explicit some of these available choices. With this enhancement, the user can program in a more complete description of the input problem using the enriched language — especially the information that might have been unknowingly glossed over because of a less expressive language.

* This research was supported in part by the NSF grant IRI-9009537.

Our goal in this paper is very modest. We do not wish to propose an entirely new semantics of inheritance, but instead, extend the semantics given in [8] to enable making subtle distinctions among networks that would not have been possible otherwise. Thus, instead of having traditional networks and a number of different but related inheritance theories, we propose to amalgamate harmoniously a family of these theories in order to enable more complete and satisfactory representation. In other words, in place of having several different monochromatic pictures each having a different color, we wish to develop an integrated framework that will permit painting many multi-coloured pictures.

In Section 2, we motivate the different expressive features to be incorporated in the enhanced inheritance networks. In Section 3, we describe, in detail, the formal syntax and semantics of annotated inheritance networks (abbreviated as AINs in the sequel). We then briefly discuss issues related to efficient implementation. We summarize our conclusions in Section 4.

2 Motivation for Expressive Enhancements

We motivate the features supported by our AINs through examples.

A word about the notation used in Figure 1. Ordinary arrows stand for defeasible arcs, while the bold arrows stand for strict arcs. Bidirectional arrows support contrapositive reasoning, while the unidirectional arrows do not. The end of the arrow marked with "–" refers to the negation of the property, while the unmarked end is interpreted as referring to the property. See Section 3 for a detailed explanation.

- *Specificity Relation*: Inheritance conflicts in networks that support both multiple inheritance and exceptions can be resolved using class-subclass relationships implicit in the network. For example, typically, mammals are not aquatic. Whales are aquatic mammals. Given that Shamu is a whale, we conclude that Shamu is aquatic. This is because Shamu being a whale provides more specific information than it being a mammal does. See Figure 1(a).
  There are a number of different semantics of INs that primarily differ in the implicit specificity they infer from the topology of the network. The inheritance theories in [9] determine specificity by computing inheritance information. For example, if $r(n)$ and $q(n)$ hold, and q's are p's and r's are $\neg p$'s, and r inherits q, then r is more specific than q, and hence, n inherits $\neg p$ via r over p via q. This notion of local specificity (denoted as $\prec$) can be visualized as ordering the in-arcs into a property node. See Figure 1(b).
- *Preferential Inheritance*: Preferential networks allow representation of certain unambiguous situations whose traditional representation resembles an ambiguous network [7]. This extension can be integrated with our theories by imposing additional preferential constraints on the ordering of the in-arcs (also denoted as $\prec$) into property nodes. For example, typically, undergraduate students are unemployed, while undergraduate teachers are employed. However, undergraduate teaching assistants are employed. Thus,

even though the IN representing these facts resembles a Nixon diamond, the ambiguity about the employment status of undergraduate teaching assistants can be resolved in favor of inheriting *employed through undergraduate teachers* over inheriting *unemployed* through *undergraduate students*. See Figure 1(c).

- *Strict and Defeasible arcs*: It is useful to distinguish between strict and defeasible arcs [6]. For example, typically, native speakers of German are not born in America. Persons born in Pennsylvania are born in America. Given that Hermann is both a native speaker of German and is born in Pennsylvania, we can conclude that he is born in America. In this example, the *strict* conclusion that Hermann is born in America overrides the *defeasible* conclusion that Hermann is not born in America. See Figure 1(d).

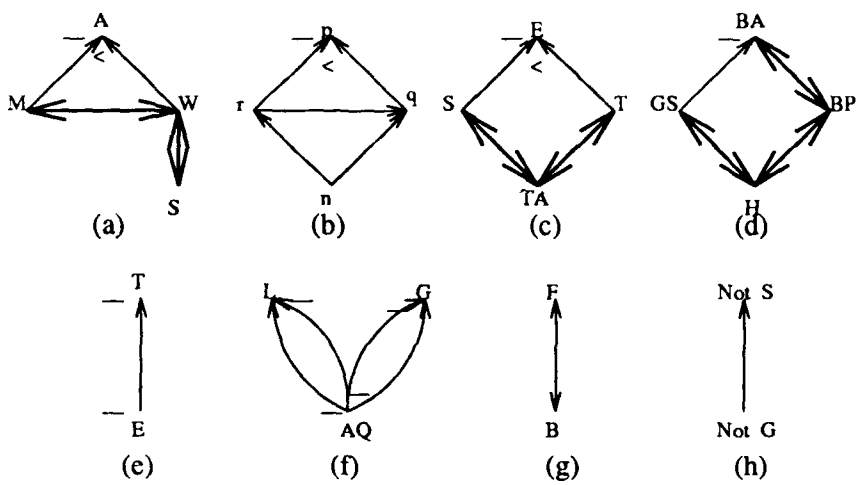


Fig. 1. Examples

- *Skeptical vs Credulous Interpretation of Ambiguity*: In the presence of conflicting evidence supporting whether or not n is a p, there are potentially two possible ways to interpret ambiguity. In the credulous case, we arbitrarily pick one of the two conclusions — $p(n)$ or $\neg p(n)$, and proceed from there; while, in the skeptical case, we eschew from drawing any conclusion about the p-ness of n. For example, according to defendent Judge Clarence Thomas's testimony, he was not guilty, while, according to accuser Professor Anita Hill's testimony, he was guilty. Typically, the guilty is punished, while the innocent is acquitted. In this ambiguous situation, the Senate vote was used to pick the "credulous" conclusion that acquitted Judge Thomas. In contrast, consider the example of a chinese graduate student in a U.S. university. Typically, chinese students are weak in English, while graduate students in US universities are strong in English. In the context of deter-

mining whether to award a teaching assistantship or to award a research assistantship to such a student, we prefer to remain "skeptical" in order to further investigate the person's proficiency in English, rather than arbitrarily choose an inappropriate conclusion.

- *Symmetric Arcs*: In INs, a positive (resp. negative) defeasible arc from p to q propagates only those individuals to q (resp. $\neg$q) that possess p positively provided that they are not exceptional. We generalize these arcs to support propagation of individuals that possess property p negatively, to inherit property q positively or negatively. This extension makes the language constructs more symmetrical, enabling representation of such statements as: Unemployed persons do not pay taxes. (See Figure 1(e).) Land-dwellers have lungs and no gills. Aquatic animals have gills but no lungs. (See Figure 1(f).)
- *Contraposition*: A number of defaults can be contraposed in the same way as strict rules. For example, typically, birds fly, and, typically, non-flying objects are not birds. However, there are cases where it is counter-intuitive to contrapose defaults. For instance, if it can be shown that someone is not guilty then one can conclude that the same person is not a suspect. But, if someone is a suspect, it does not follow that the person is guilty. Similarly, people are normally not diabetic, but diabetics are people. See Figure 1(g)(h).

As can be observed, there is no single formalism in the literature that can represent all these examples. The formalism we develop in the sequel can satisfactorily represent all the examples given above in a unified framework.

3 Annotated Inheritance Networks

3.1 Language

We extend the syntax of INs to AINs to incorporate features described above.

Definition 1. An *annotated inheritance network* is an ordered directed acyclic graph consisting of

- a set of nodes that can be partitioned into two sets — the set of individual nodes **I** and the set of property nodes **P**;
- a boolean function cred_skep : **P** $\mapsto$ boolean, which specifies whether a credulous or a skeptical meaning of the node is desired;
- a set of arcs **A** and a labelling function arctype: **A** $\mapsto$ ($\{\overset{++}{\Longleftrightarrow}, \overset{+-}{\Longleftrightarrow}, \overset{-+}{\Longleftrightarrow}, \overset{--}{\Longleftrightarrow}\} \cup \{\overset{++}{\longrightarrow}, \overset{+-}{\longrightarrow}, \overset{-+}{\longrightarrow}, \overset{--}{\longrightarrow}\} \cup \{\overset{++}{\longleftarrow}, \overset{+-}{\longleftarrow}, \overset{-+}{\longleftarrow}, \overset{--}{\longleftarrow}\}$);
- for each node p $\in$ **P**, an asymmetric preferential inheritance relation $\prec_p$ on its in-arcs.

Informally, for a property node p, if cred_skep(p) holds then we associate a credulous meaning with p, else we associate a skeptical meaning with p. The arc labelled $\overset{++}{\Longleftrightarrow}$ (resp. $\overset{+-}{\Longleftrightarrow}, \overset{-+}{\Longleftrightarrow}, \overset{--}{\Longleftrightarrow}$) from node p to node q represents a *strict* arc from p to q (resp. from p to $\neg$q, from $\neg$p to q, from $\neg$p to $\neg$q).

Similarly, the arc labels in $\{\xrightarrow{++}, \xrightarrow{+-}, \xrightarrow{-+}, \xrightarrow{--}\}$ represent *defeasible* arcs that *do not support contraposition*, and the arc labels in $\{\xleftarrow{++}, \xleftarrow{+-}, \xleftarrow{-+}, \xleftarrow{--}\}$ represent *defeasible* arcs that *do support contraposition*. In the literature, there are at least two different interpretations of contraposition with respect to defeasible arcs. In [15], the defeasible arcs are oriented and the forward conclusions override the conflicting contrapositive conclusions, whereas, in [4], the defeasible arcs are truly symmetrical with respect to contraposition. So, if we subscribe to the former view, the contraposable arcs can be oriented from bottom to top or from left to right. We also require that, if the source node of an arc is an individual node in **I**, then the arc is strict and the arctype must be one of $\overset{++}{\Longleftrightarrow}$ or $\overset{+-}{\Longleftrightarrow}$. The *sign* of the source node of the arcs labelled $\overset{++}{\Longleftrightarrow}, \overset{+-}{\Longleftrightarrow}, \xrightarrow{++}, \xrightarrow{+-}, \xleftarrow{++}, \xleftarrow{+-}$ is positive, and the *sign* of the destination node of the arcs labelled $\overset{+-}{\Longleftrightarrow}, \overset{--}{\Longleftrightarrow}, \xrightarrow{+-}, \xrightarrow{--}, \xleftarrow{+-}, \xleftarrow{--}$ is negative. If the sign is missing, it is assumed to be $+$.

3.2 Semantics

We specify the semantics of AIN in two parts — the strict part and the defeasible part. Note that a strict conclusion propagates through a strict arc as a strict conclusion, while it propagates through a defeasible arc as a defeasible conclusion. In contrast, a defeasible conclusion propagates through all arcs as a defeasible conclusion.

Let Γ be an AIN and x, y, ...denote the nodes in Γ. In the sequel, the notation of the form $x \overset{++}{\Longleftrightarrow} z$, $x \xrightarrow{++} z$ etc. refers to a path from x to z rather than a direct arc. Whenever we want it to refer to an arc from x to z, we will say so explicitly.

To define inheritance, we make rigorous the notion of a *supported* path. Informally, a path containing all strict arcs is supported. In Figure 1(d), the strict path from node H to node BA via node BP, is supported. However, not all defeasible paths are supported. In Figure 1(b), the defeasible path from node N to node P via node Q, which provides some evidence in support of N possessing P, is not supported, because it is overriden by the conflicting defeasible path from node N to node P via node R. To determine defeasible paths and to resolve conflicts among them, we formalize the auxiliary notions of *has_evidence_for* and of *defeat*. Finally, the definition of *supports* for defeasible paths deals with the interpretation of ambiguity.

We introduce the *supports* relation, denoted as $\triangleright$, below.

Definition 2. $\Gamma \triangleright x \overset{++}{\Longleftrightarrow} y$ if one of the following holds:
- (direct) $x \overset{++}{\Longleftrightarrow} y$ is an arc.
- (forward) $\Gamma \triangleright x \overset{++}{\Longleftrightarrow} z$, and $z \overset{++}{\Longleftrightarrow} y$ is an arc.
- (forward) $\Gamma \triangleright x \overset{+-}{\Longleftrightarrow} z$, and $z \overset{-+}{\Longleftrightarrow} y$ is an arc.
- (contrapositive) $\Gamma \triangleright x \overset{++}{\Longleftrightarrow} z$, and $y \overset{--}{\Longleftrightarrow} z$ is an arc.
- (contrapositive) $\Gamma \triangleright x \overset{+-}{\Longleftrightarrow} z$, and $y \overset{-+}{\Longleftrightarrow} z$ is an arc.

Similarly, we can define $\Gamma \triangleright x \overset{+-}{\Longleftrightarrow} y$ (resp. $x \overset{-+}{\Longleftrightarrow} y$, $x \overset{--}{\Longleftrightarrow} y$).

Note that even though in the real world there cannot exist an x that possesses both y and $\neg y$, there can be still be such inconsistencies in the input supplied by the user to the inheritance reasoner. In such situations, we tolerate and localize its effect [2] [18].

In order to specify defeasible conclusions, we need to explain conflict resolution strategies for disambiguation. Briefly, we use the following criteria: (1) A strict conclusion always overrides the corresponding conflicting defeasible conclusion. (See Figure 1(d) in Section 2.) (2) A more specific defeasible conclusion overrides a corresponding less specific conflicting defeasible conclusion. (See Figure 1(a) in Section 2.) (3) Furthermore, in some approaches, a "forward" defeasible conclusion overrides a conflicting "contrapositive" defeasible conclusion [8] [15] [3], while in others, the contrapositive defeasible arcs are truly symmetrical [4].

We explain informally the specificity relationship used in [9] to resolve conflicts among defeasible conclusions. See Figure 1(b). If r inherits q, then for all nodes n, and for all property nodes p which are parents of q and r, the inheritance of $\neg$p by n via r dominates over conflicting inheritance of p by n via q. This is denoted by making the arc from q to p $\prec_r$ the arc from r to p. Similarly, if $\neg$r inherits q, then for all nodes n, and for all property nodes p, which are parents of q and r, the inheritance of p by n by virtue of being a $\neg$r dominates over conflicting inheritance of p by n by virtue of being a q. Observe also that the specificity relation and the inheritance relation are defined *mutually recursively*.

To generalize local specificity, we first formalize the *defeat* relation. Figure 2 shows all the possibilities that can arise for the AIN (where $\prec_p$ indicates the relative strength of inheritance of p through the arcs).

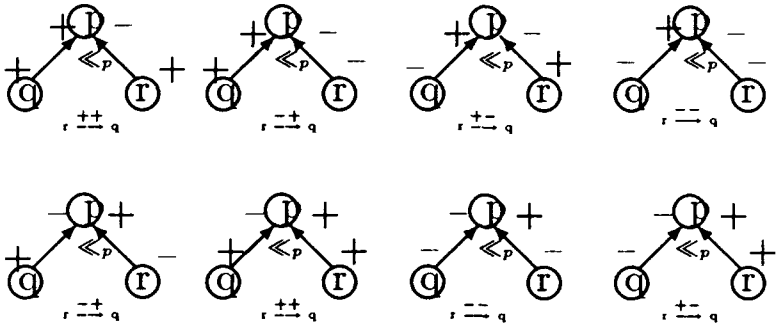


Fig. 2. Implicit Specificity

Informally, an arc $y \xrightarrow{++} z$ is defeated for x (that possesses y) if there is more specific evidence that prohibits x from propagating further to node z through the arc $y \xrightarrow{++} z$. Formally,

Definition 3. Γ *defeats* the arc $y \xrightarrow{++} z$ for x if $(\Gamma \rhd x \overset{++}{\Longleftrightarrow} y)$ or $(\Gamma \rhd x \xrightarrow{++} y)$

and if any one of the following holds:
- *(Strict overrides defeasible.)* $\Gamma \rhd x \overset{+-}{\Longleftrightarrow} z$.

- *(Conflict resolution using Specificity, Preferential Inheritance.)*

  - there exists an arc $w \overset{+-}{\Longleftrightarrow} z$ such that $\Gamma \rhd x \overset{++}{\longrightarrow} w$, and furthermore, $\Gamma \rhd w \overset{++}{\longrightarrow} y$, or $y \prec_z w$; or
  - there exists an arc $w \overset{--}{\Longleftrightarrow} z$ such that $\Gamma \rhd x \overset{+-}{\longrightarrow} w$, and furthermore, $\Gamma \rhd w \overset{-+}{\longrightarrow} y$, or $y \prec_z w$; or
  - there exists an arc $w \overset{+-}{\longrightarrow} z$ (resp. $w \overset{+-}{\longleftarrow} z$) such that $\Gamma \rhd x \overset{++}{\longrightarrow} w$ or $\Gamma \rhd x \overset{++}{\Longleftrightarrow} w$, and furthermore, $(\Gamma \rhd w \overset{++}{\Longleftrightarrow} y)$, or $(\Gamma \rhd w \overset{++}{\longrightarrow} y)$, or $y \prec_z w$; or
  - There exists an arc $w \overset{--}{\longrightarrow} z$ (resp. $w \overset{--}{\longleftarrow} z$) such that $\Gamma \rhd x \overset{+-}{\longrightarrow} w$ or $\Gamma \rhd x \overset{+-}{\Longleftrightarrow} w$, and furthermore, $(\Gamma \rhd w \overset{-+}{\Longleftrightarrow} y)$, or $(\Gamma \rhd w \overset{-+}{\longrightarrow} y)$, or $y \prec_z w$.

Similarly, one can define the concept of *defeat* for other defeasible arcs of type $\overset{--}{\longrightarrow}, \overset{+-}{\longrightarrow}, \overset{-+}{\longrightarrow}, \overset{++}{\longleftrightarrow}, \overset{--}{\longleftrightarrow}, \overset{+-}{\longleftrightarrow}, \overset{-+}{\longleftrightarrow}$.

To define the *supports* relation for defeasible conclusions, we introduce an auxiliary relation called *has_evidence_for*, denoted as $\triangleright$. We define defeasible conclusions supported by Γ due to "forward propagation" through defeasible arcs as follows.

Definition 4. $\Gamma \triangleright x \overset{++}{\longrightarrow} z$ if any one of the following holds:
- there exists an undefeated arc $y \overset{++}{\longrightarrow} z$ (resp. $y \overset{++}{\longleftarrow} z$) wrt x and $\Gamma \rhd x \overset{++}{\Longleftrightarrow} y$.
- there exists an undefeated arc $y \overset{++}{\longrightarrow} z$ (resp. $y \overset{++}{\longleftarrow} z$, $y \overset{++}{\Longleftrightarrow} z$) wrt x and $\Gamma \rhd x \overset{++}{\longrightarrow} y$.
- there exists an undefeated arc $y \overset{-+}{\longrightarrow} z$ (resp. $y \overset{-+}{\longleftarrow} z$) wrt x and $\Gamma \rhd x \overset{+-}{\Longleftrightarrow} y$.
- there exists an undefeated arc $y \overset{-+}{\longrightarrow} z$ (resp. $y \overset{-+}{\longleftarrow} z$, $y \overset{-+}{\Longleftrightarrow} z$) wrt x and $\Gamma \rhd x \overset{+-}{\longrightarrow} y$.

Similarly, for the paths of type $\overset{--}{\longrightarrow}, \overset{-+}{\longrightarrow}, \overset{+-}{\longrightarrow}$.

We also define defeasible conclusions supported by Γ due to "backward propagation" through contrapositive arcs as follows.

Definition 5. $\Gamma \triangleright x \overset{++}{\longrightarrow} z$ if it is not the case that $(\Gamma \rhd x \overset{+-}{\Longleftrightarrow} z)$ or $(\Gamma \rhd x \overset{+-}{\longrightarrow} z)$, and if any one of the following holds:
- there exists an arc $z \overset{--}{\Longleftrightarrow} y$ and $\Gamma \rhd x \overset{++}{\longrightarrow} y$.
- there exists an arc $z \overset{-+}{\Longleftrightarrow} y$ and $\Gamma \rhd x \overset{+-}{\longrightarrow} y$.
- there exists an arc $z \overset{--}{\longleftarrow} y$ and $(\Gamma \rhd x \overset{++}{\longrightarrow} y)$ or $(\Gamma \rhd x \overset{++}{\Longleftrightarrow} y)$.
- there exists an arc $z \overset{-+}{\longleftarrow} y$ and $(\Gamma \rhd x \overset{+-}{\longrightarrow} y)$ or $(\Gamma \rhd x \overset{+-}{\Longleftrightarrow} y)$.

Similarly, for the paths of type $\overset{--}{\longrightarrow}, \overset{-+}{\longrightarrow}, \overset{+-}{\longrightarrow}$.

The supports relation for defeasible paths can be defined now.

Definition 6. For skeptical nodes p (that is, $\neg$cred_skep(p) holds):
$(\Gamma \rhd x \xrightarrow{++} y)$ **iff** $(\Gamma \triangleright x \xrightarrow{++} y)$ **and not** $(\Gamma \triangleright x \xrightarrow{+-} y)$
$(\Gamma \rhd x \xrightarrow{+-} y)$ **iff** $(\Gamma \triangleright x \xrightarrow{+-} y)$ **and not** $(\Gamma \triangleright x \xrightarrow{++} y)$
For credulous nodes p (that is, cred-skep(p) holds):
$(\Gamma \rhd x \xrightarrow{++} y)$ **iff** $(\Gamma \triangleright x \xrightarrow{++} y)$ **and not** $(\Gamma \rhd x \xrightarrow{+-} y)$
$(\Gamma \rhd x \xrightarrow{+-} y)$ **iff** $(\Gamma \triangleright x \xrightarrow{+-} y)$ **and not** $(\Gamma \rhd x \xrightarrow{++} y)$
Similarly, for $\xrightarrow{-+}$, $\xrightarrow{--}$.

Definition 7.
x inherits y **iff** $(\Gamma \rhd x \xrightarrow{++} y)$ or $(\Gamma \rhd x \xLeftrightarrow{++} y)$.
x inherits $\neg y$ **iff** $(\Gamma \rhd x \xrightarrow{+-} y)$ or $(\Gamma \rhd x \xLeftrightarrow{+-} y)$.
Similarly, for $\neg x$ inherits y and $\neg x$ inherits $\neg y$.
An *expansion* of AIN Γ is a minimal inherits relation containing arcs in Γ.

From a practical standpoint, we identify a rich subset of AINs (that properly includes INs) for which an expansion is "easy" to compute. (For other AINs, the computation of an expansion may require other book-keeping overheads as described in the next section.) The sufficient condition we wish to propose restricts the propagation of contrapositive defeasible conclusions.

Definition 8. An AIN is *bounded* if it does not contain arc pairs like $x \xrightarrow{++} y$ and $x \xleftarrow{-+} z$ (resp. $x \xLeftrightarrow{-+} z$), $x \xrightarrow{-+} y$ and $x \xleftarrow{+-} z$ (resp. $x \xLeftrightarrow{+-} z$), $x \xrightarrow{--} y$ and $x \xleftarrow{++} z$ (resp. $x \xLeftrightarrow{++} z$), $x \xrightarrow{+-} y$ and $x \xleftarrow{-+} z$ (resp. $x \xLeftrightarrow{-+} z$), etc.

The following results can also be proven along the lines of similar results shown for INs in [9].

Theorem 9. *Every bounded AIN has an expansion. Furthermore, if every node in the bounded AIN is skeptical, then the AIN admits a unique expansion.*

3.3 Implementation

The above specification defines an *expansion* of a bounded AIN. To compute it efficiently non-monotonic revision of conclusions should be minimized. That is, we do not assert x inherits y if a stronger evidence for x inherits $\neg y$ can potentially be uncovered. To avoid unnecessary overheads we compute the meaning monotonically, by asserting only the final conclusions, and not the intermediate revisable ones. To this end, we constrain the control flow of the inheritance algorithm using the criteria to resolve conflicts. In particular, we compute all strict conclusions before any defeasible conclusion. To compute defeasible conclusions, we process the nodes bottom-up to let more specific evidence dominate less specific ones. Finally, we block conflicting contrapositive conclusions [15] [8].

A sketchy exposition of the inheritance algorithm for bounded AINs follows: It consists of two phases. The first phase computes strict conclusions, while,

the second phase computes defeasible conclusions. This allows strict conclusions to override defeasible conclusions. The first phase consists of a number of passes through the AIN to propagate contrapositive strict conclusions. The second phase consists of only two passes for bounded AINs — an bottom-up pass and a top-down pass. In the former, we compute the "forward" conclusions, while in the latter, we compute the "contrapositive" conclusions [15] [8]. For bounded AINs, only two passes are sufficient to compute an expansion because the boundedness condition prohibits indiscriminate contrapositive propagation [9]. However, for arbitrary AINs, multiple passes may be required. Furthermore, this may necessitate nonmonotonic revision until the meaning "stabilizes".

The computation of an expansion of bounded AIN requires polynomial-time. The phase one requires multiple passes of the AIN but the number of passes is bounded by the length of the longest path in the network. The phase two can be done by generalizing the polynomial-time two-pass inheritance algorithm for INs given in [9].

4 Discussion and Conclusion

We have proposed *expressive enhancements* to INs that enable a knowledge engineer to make explicit information present in the input, but that cannot be represented using the INs. We believe that the proposed extensions will benefit the user because it will permit the user to decide what knowledge can be encoded in the system, and give the user understandable formal guarantees about the quality of the conclusions that will be generated [1].

In contrast with the path-based approaches discussed in [11] [20], our approach is "conclusion"-based. That is, an expansion is a collection of conclusions supported by the AIN, and not a set of permitted paths. Thus, we do not have the counterpart for *floating conclusions* and *zombie paths* of [11], and *reinstators* of [20].

We now briefly touch upon some of the examples given in [20]. Consider the *wild-chicken example* of [20]. Wild-chickens are chickens, and chickens are birds. Chickens have weak-wings, while birds and wild-chickens have strong-wings. Entities with weak-wings do not fly, while entities with strong-wings do fly. This can be represented in our formalism as shown below: $\{WC \overset{++}{\Longleftrightarrow} C$, $C \overset{++}{\Longleftrightarrow} B$, $WC \overset{+-}{\longrightarrow} WW$, $C \overset{++}{\longrightarrow} WW$, $B \overset{+-}{\longrightarrow} WW$, $WW \overset{-+}{\longrightarrow} F$, $WW \overset{+-}{\longrightarrow} F\}$. Our theory supports the following conclusions which are intuitively satisfactory: If x is a bird, then it has strong wings and it flies. If x is a chicken, then it has weak wings and it does not fly. If x is a wild chicken, then it has strong wings and it flies.

The differences between the specificity espoused here and that in [5] is described in [9] [10]. One can further extend AINs to indicate disjoint pairs of nodes; to specify when the children of a node are exhaustive [12]; or to tag each conclusion explicitly with the class name to help in conflict resolution.

Acknowledgement

I would like to thank Yaqiong Zhang for going through an earlier draft of this paper.

References

1. F. Bacchus, A modest, but semantically well-founded, inheritance reasoner, In *IJCAI-89,* pp. 1104-1109, 1989.
2. N. Belnap, How a computer should think, in: G. Ryle (ed.), *Contemporary Aspects of Philosophy* (Oriel Press, 1977) 30–56.
3. H. Geffner, Defaults Reasoning: Causal and Conditional Theories, *Ph.D. Dissertation*, University of California at Los Angeles, 1989.
4. M. L. Ginsberg, A local formalization of inheritance: preliminary report, 1988.
5. J. Horty, R. Thomason, and D. Touretzky, A skeptical theory of inheritance in nonmonotonic semantic networks, *Artificial Intelligence,* **42** (1990) 311-348
6. J. Horty and R. Thomason, Mixing strict and defeasible inheritance, In *AAAI-88*, pp. 427-432, 1988.
7. T. Krishnaprasad, M. Kifer, and D. S. Warren, On the declarative semantics of inheritance networks, In *IJCAI-89,* pp. 1093-1098, 1989.
8. Krishnaprasad Thirunarayan, The semantics of inheritance networks, *Ph.D. Dissertation*, State University of New York at Stony Brook, 1989.
9. Krishnaprasad Thirunarayan, Implementation of an efficient inheritance reasoner, *Technical Report WSU-04-91*, Wright State University, 1991.
10. Krishnaprasad Thirunarayan and M. Kifer, A theory of nonmonotonic inheritance based on annotated logic, In *Artificial Intelligence* **60** (1993).
11. D. Makinson and K. Schlechta, Floating conclusions and zombie paths: two deep difficulties in the "directly skeptical" approach to defeasible inheritance nets, In *Artificial Intelligence* **48** (1991) 199–209.
12. E. Neufeld, Defaults and probabilities; extensions and coherence, In *KR-87*, pp. 312-323, 1989.
13. L. Padgham, Negative reasoning using inheritance, In *IJCAI-89,* pp. 1086-1092, 1989.
14. J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988.
15. H. Przymusinska and M. Gelfond, Inheritance hierarchies and autoepistemic logic, University of Texas at El Paso, 1989.
16. B. Selman and H.J. Levesque, The tractability of path-based inheritance, In *IJCAI-89,* pp. 1140-1145, 1989.
17. L. A. Stein, A preference-based approach to inheritance, Brown University, 1990.
18. R. Thomason, J. Horty, and D. Touretzky, A calculus for inheritance in monotonic semantic nets, In *ISMIS-87* pp. 280–287, 1987
19. R. Thomason and J. Horty, Logics for inheritance theory, In *Non-Monotonic Reasoning,* M. Reinfrank et al (eds.), 1989.
20. D. Touretzky, R. Thomason, and J. Horty, A skeptic's menagerie: conflictors, preemptors, reinstaters, and zombies in nonmonotonic inheritance. In *IJCAI-91,* pp. 478-483, 1991.
21. D. Touretzky, The mathematics of inheritance systems, Morgan Kaufmann, 1986.
22. D. Touretzky, J. Horty, and R. Thomason, A clash of intuitions: the current state of nonmonotonic multiple inheritance systems, In *IJCAI-87,* pp. 476-482, 1987.

A Connectionist-Symbolic Cognitive Model

Guilherme Bittencourt

Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais
Caixa Postal 515 - CEP 12.201-097 - São José dos Campos - SP - Brazil
E-mail: inpelac@brfapesp.br

Abstract. This paper describes a formal model for the cognitive activity, which is an instantiation of a more general proposal for a research line in artificial intelligence. The main contribution of the paper is the specification of a wave propagation model that performs inference in predicate logic without quantifiers through an interference mechanism. The model is highly parallel, and is flexible enough to be extended, in a natural way, to simulate first-order logic, fuzzy logic, four-valued logic, and uncertain reasoning. The model is part of an architecture integrating neural networks and symbolic reasoning to simulate cognitive activities.

1 Introduction

Artificial intelligence could be characterized as a collection of techniques adapted to solve specific problems, even the fundamental research being divided among several approaches, e.g. the physical symbol systems model (Newell and Simon, 1976) and the connectionist model (such as Rumelhart and McClelland, 1986a and 1986b).

In a special volume of the *Artificial Intelligence Journal* on Foundations of Artificial Intelligence, Kirsh (1991a) identifies five issues which have become focal points of debate in the field, and serve as dividing lines of positions:

- Pre-eminence of knowledge and conceptualization
- Disembodiment
- Language-like structure of the kinematics of cognition
- Possible independence of learning and cognition
- Uniform architecture

This paper describes a formal model for the cognitive activity, which is an instantiation of a more general proposal for a research line in artificial intelligence. The arguments underlying this proposal are the following: (i) the only known intelligent being is the result of an evolutive process modelled by the principles of *natural selection*, (ii) the physical entity that embodies human intelligence, the brain, is built out of matter that can be modelled by *physical theories*.

Based on these arguments, we propose the application of the following two basic principles to guide simulations of the cognitive activity:

Principle 1 *The cognitive activity should be simulated through a model epistemologically (McCarthy and Hayes, 1969) compatible with the theory of evolution.*

Principle 2 ***The cognitive activity should be based on the interaction of a large number of functionally independent unities of a few types (Changeux, 1983). The communication between different unities should be simulated through the emission and reception of information waves.***

According to the above principles, the proposed research line implies the following positions with respect to the above foundational issues:

- Concepts are fundamental to the cognitive capacity and take the form of stationary waves in a medium characterized by a particular structure of independent unities. These waves are not always active, but can be stored under the form of temporal sequences of specific wave frequencies. Memory can be explained as the propagation and interference pattern of a set of such waves, activated by some input flow of information.
- Perception and motor control are the basic mechanisms for the formation of particular wave patterns which constitute a concept for each cognitive agent. The apparent similarity of concepts acquired by cognitive agents is a consequence of the regularities of the physical world and cultural interaction, the lower levels of cognition being specific to each agent.
- Language is the basis for the creation of wave patterns associated to abstract concepts. Without language the cognitive ativity is restricted to correlations of different senses and their significance to the survival of the living agent.
- Learning and cognition are closely related. Between the levels of sensory perception and abstract language there are several levels of complexity. The kinematics of cognition are similar in both extreme levels, but the intermediate levels are particular to each agent and built through learning activity according to natural selection mechanisms.
- There is a single architecture underlying all cognition, but only in the sense that the building blocks are similar. Cognition is mostly a dynamic property of the physical brain. The same concept is stored in totally different wave patterns in different cognitive agents.

The point of this paper is to introduce a simple formal model coherent with the above arguments. This model intends to simulate a cognitive agent that communicates with the external world, stores information about the environment, makes inferences using the available information, and learns to generate adequate reactions according to the state of the environment. The cognitive agent is modelled as a community of formal cognitive organisms whose behavior is defined according to the natural selection principles.

The model associated to each cognitive organism consists of two levels: (i) the categorization level, which corresponds to the communication process between the environment, the memory and the propagation level, and (ii) the propagation level, which corresponds to the reasoning process. The formalism used to define the categorization level is neural network theory. The propagation level is defined as a theorem prover, based on a wave propagation formalism, acting on the information provided by the categorization level.

The main contribution of this paper is the specification of the wave propagation model that performs inference in predicate logic without quantifiers through an

interference mechanism. This model is highly parallel, and is flexible enough to be extended, in a natural way, to simulate first-order logic, fuzzy logic, four-valued logic, and uncertain reasoning. A further contribution is the introduction of an architecture integrating neural networks and symbolic reasoning to simulate cognitive activities.

The paper is organized as follows: In Section 2, the categorization level is sketched. In Section 3, the propagation level and its logical interpretation are defined and its behavior is analized. Afterwards, we describe a control mechanism, based on natural selection principles, and, in Section 5, we comment upon some possible extensions to the model. In Section 6, the proposed model is compared with some other proposals in the literature. Finally, in Section 7, we present some conclusions and discuss future directions for research.

2 Categorization

The functions of the categorization level are: classification of information received from the environment, storage/retrieval of information into/from the memory, and transmission of information to the propagation level.

To formalize the environment, we supose the availability of information patterns about the world, and evaluation functions. These functions can be used to evaluate a given information vector according to its usefulness to the cognitive organism's goals (Newell e Simon, 1976). More formally:

Given an arbitrary environment E, and the set of information vectors of N bits I_N, we define the mapping γ:

$$\gamma : E \rightarrow I_N$$

This mapping corresponds to sensory functions. For present purposes, we consider only the information vectors; any further detail about sensory mechanisms are beyond the scope of the paper.

Let n be the cardinality of the predicate set P, which is used in the propagation level, and $\mathbf{R}$ be the real numbers. We introduce the evaluation function ρ:

$$\rho : (I_N)^n \rightarrow \mathbf{R}$$

This function can give an a priori evaluation to each possible situation, and is a model for the adaptation mechanisms – such as fear, hunger, sexual attraction, etc.

The communication between categorization and propagation levels is achieved through the association of each information pattern available from the environment, with an element of some Herbrand universe:

$$\mu : I_N \rightarrow H$$

We define next the function ϕ from the set H to the natural numbers:

$$\phi : H \rightarrow \mathbf{N}$$

This function defines a possible order on the Herbrand universe, corresponding to an order of the information patterns, and is modelled by a neural network (Smolensky, 1991) able to associate a natural number to each information pattern. Initially, the neural network classifies the information according to the temporal order.

Given a vector ι in I_N, this information is transmitted to the propagation level in the form of n waves, one for each predicate in P. The propagation space, according to the definition in the next section, is an n-dimensional space where each axis is associated to a characteristic frequency, representing a predicate. Each wave initiates its propagation from a predicate axis, with the characteristic frequency associated to this axis. The point on the axis corresponds to the number associated to the information pattern through the mappings ϕ and μ, i.e. the number $\phi(\mu(\iota))$.

Let W be a set of points in the propagation space associated to a set of n-tuples of information vectors $\Gamma = \{\iota_1, ..., \iota_k\}$, such that $\sum_i \mid \rho(\iota_i) \mid$ presents a high value. Let also the points in W be activated in a given order with the frequencies $F = \{f_1, ..., f_k\}$.

Memory is modelled by a set of such frequency sequences, classified according to their value for $\sum_i \mid \rho(\iota_i) \mid$. As the activation frequency of a point depends on the frequencies of the waves that activated it, it is always possible to determine the predicates involved in the activation. Thus, the frequency sequences contain all the information necessary to repeat the propagation phenomenon.

To minimize the propagation time, it is interesting that the distance between points associated to related information vectors be small. This can be achieved through the use of the memory information as a learning feedback to the neural network that classifies the information.

If the information flow stops, e.g. during sleep periods, the available time can be used to compare and combine the frequency sequences in the memory. Without environmental information, an arbitrary internal time can be imposed on the propagation mechanism, and the previous experiences can be repeated in a controlled way. It is also possible to refine the consequences of a given frequency sequence, without the perturbation of the environment reactions. Analogous frequency sequences can be combined to form graphs, where different possibilities are associated to each other in a single representation of possible choices.

3 Propagation

The propagation level is a simple formal model to perform inference in a predicate logic without quantifiers, through a wave propagation mechanism. The input information is delivered by the categorization level, and is modelled as an ordered Herbrand universe. The output information is send back to the categorization level, and is associated to reactions of the cognitive agent.

Consider the following logical system. Let P be a set of n predicate symbols, and F be a set of function (and constant) symbols. Let H be the Herbrand universe constructed with the set F, and C be the set of all closed logical clauses that can be formed with the predicates in P and the expressions in the set H. Each element in C corresponds to a set of closed literals, and is called a *clause*. A clause can be semantically interpreted as a conjunction of literals, the usual representation adopted in the literature of automatic theorem proof, or alternatively as a disjunction of literals.

Given a closed logical expression, it can always be represented in two different canonical forms: a disjunction of conjuntions of literals (the usual form), or a con-

junction of disjunctions of literals. Each canonical form can be represented as a set of elements of C, i.e. a set of clauses.

A clause is a contradiction, if it contains two identical literals with different signs. If the interpretation of the representation is a disjunction of conjunctions, the fact that one clause is a contradiction implies that all the representation is a contradiction. If the interpretation is a conjunction of disjunctions, the contradictory clause can simply be eliminated.

A clause *subsumes* another clause if it is a subset of the other clause. In both canonical representations clauses that are subsumed by other clauses can be eliminated.

Given one of the canonical representations of a logical expression, it is possible to obtain the other through the distributivity property of the logical operators *and* and *or*.

The proposed wave propagation model is based in an original logical result that, given any logical expression represented in the usual canonical form, the transformation of this representation into the other canonical form, followed by the elimination of the subsumed clauses, and the reverse transformation back to the original form, performs automatically all the possible logical inferences by resolution allowed by the original representation. This result, and its extension to first-order logic is presented in Bittencourt (1993).

The problem with this method is that it is computationally expensive (distributivity is exponential), but if the adequate representation for clauses is chosen it can be performed in parallel.

We propose a representation that associates each clause in a logical system to a point in an n-dimensional discrete Euclidean space. First, we recall the definition of the mapping ϕ from the set H to the natural numbers:

$$\phi : H \rightarrow \mathbf{N}$$

We also define a total order for the predicate set P.

$$\eta : P \rightarrow \{0, ..., n-1\}$$

We can now define a mapping from the clause set C to n-tuples of natural numbers.

$$\psi : C \rightarrow \mathbf{N}^n$$

$$\psi(L_0 \vee ... \vee L_n) = (z_0, ..., z_n)$$

Where, for all $x \in H$:
if $L_i = P_i(x)$ then $z_i = \phi(x)$,
if $L_i = \neg P_i(x)$ then $z_i = -\phi(x)$,
if $L_i = False$ then $z_i = 0$, and
$i = \eta(P_i)$.

In this notation, the conversion from a canonical representation into the other can be performed through simple vectorial transformations of the points representing the clauses. These vectorial transformations can be executed in parallel through the propagation and interference of information waves, if these waves have the appropriate frequencies (which are associated to the predicates present in the clause that originated the wave). Besides that, a subsumed clause lies always in a superspace generated by the subsuming clause point along the other dimensions of the predicate space. Using this property, all subsumed clauses can be eliminated by an inhibiting wave propagating from each clause point in the direction of its superspaces.

4 Control

The model of the control mechanism is based on the application of the principles of natural selection. These principles are the following: (i) perpetuation possibility, (ii) limited resources, and (iii) sufficient variation to generate evolution. To apply these principles, it is necessary to identify, in the model, what corresponds to the organisms, the environment, the resources, and the perpetuation and variation mechanisms. A formal cognitive organism consists of the interconnection of a propagation mechanism, a neural network classifier, and a memory.

The *environment* consists of the flow of information provided by the senses of a cognitive agent. This flow of information is received by a community of cognitive organisms. Each organism processes the information and reacts according to the result. If the reactions are adapted to the environment, the organism receives a good evaluation. This evaluation can be thought of as the adaptability of the cognitive agent, for it determines what are its necessities and weaknesses.

The *limited resource* is time. The cognitive organisms that are able to generate a relevant answer to a given situation quicker than the others, are considered better adapted to the specific environment. As long as the answers generated by the organisms are present in the environment, they can influence other organisms, allowing the possibility of evolutive phenomenona such as symbiosis, parasitism, etc.

The main *perpetuation and variation mechanism* is the exchange of experiences between two independent cognitive agents.

The natural selection process can be simulated, according to the previous definitions, by determining the sequence of available information and the evaluation function. The internal behavior of each cognitive organism corresponds to the definitions in the previous sections. The total population of cognitive agents simulates a cognizer acting in the real world.

5 Extensions

A first possibility of extension to the propagation model is to use the geometric characteristics of the adopted notation to investigate the global properties of a set of clauses. Topological properties of these sets – such as symmetry, dispersion, relative distances, etc – can be used as a guide to structure the memory information and to classify the situations at the categorization level. This can be the basis for an analogy reasoning mechanism.

Another possibility is to extend the formalism to first-order logic. Quantified expressions can be represented by stationary waves associated to surfaces in the propagation space. These surfaces can be represented analitically, and their interference can be expressed by an algebraic transformation. This fact makes the use of computer algebra tools, e.g. Hearn (1987), interesting to analyse the wave behavior.

The fact that the model is parameter dependent allows a further extension, which consists of allowing a wider range of values for each parameter, generalizing the logic *concepts associated to them*. For example, modification of the propagation rules allowing contradictory clauses generalizes the formalism to logics with more than two truth values (Belnap, 1977; Patel-Schneider, 1985). Continuous logical values between *false* and *true*, together with the introduction of uncertain information propagation rules, allow the extension of the formalism to fuzzy logic (Zadeh, 1975).

In the categorization model, it is stated that the classification of information patterns should be modelled through neural networks, but the architecture of these neural networks and exactly how they connect to the memory is yet to be defined. The specification of such architectures is a necessary extension to the categorization model.

A further extension direction would be to define a categorization level based on a symbolic approach. Knowledge representation tools, e.g. MANTRA (Bittencourt, 1989 and 1990), are well adapted to simulate the necessary functions at the categorization level.

6 Discussion

In this section, we compare some approaches to the problem of simulating cognition presented in the literature with the proposed model. This discussion does not intend to be exaustive, as it covers only some proposals closely related to the subject of the paper. Detailed descriptions of the methaphors of the mind can be found in Hampden-Turner (1981), and a discussion of the relationship of Artificial Intelligence and Philosophy can be found in Torrance (1984).

Presenting his view of the "logical approach" to Artificial Intelligence, Nilsson (1991) proposes the following three theses:

Thesis 1 *Intelligent machines will have knowledge of their environments.*

Thesis 2 *The most versatile intelligent machines will represent much of their knowledge about their environment declaratively.*

Thesis 3 *For the most versatile machines, the language in which declarative knowledge is represented must be at least as expressive as first-order predicate calculus.*

The proposed model supports the first of these theses with no restrictions, and proposes an economic way of storing the knowledge. If this storing mechanism is declarative depends on the interpretation of the word "declarative". What is proposed is a representation closely related to the acquisition and retrival processes. This representation stores information in a structural form adapted to analogical reasoning. In this sense, it is a declarative representation. Although the proposed

formal model does not cover first-order logic, it is possible to extend it to first-order logic and beyond. Another intrinsic characteristic of the model is that it has a temporal behavior not present in first-order logic, but fundamental to cognitive agents living in a changing environment.

In his response to Nilsson's paper, Birnbaum (1991) considers as the main problems of the logical approach the emphasis on sound, deductive inference, and the tendence to ignore other sorts of reasoning such as probabilistic reasoning, reasoning from examples or by analogy, and reasoning based on the formation of faulty but useful conjectures and their subsequent elaboration and debugging, besides the presumption that model-theoretic semantics is central to knowledge representation. The proposed model reconciliates the two positions: it presents the possibility of probabilistic, analogical and conjectural reasoning in a framework defined from Herbrand's Theorem as a starting point, precisely what Birnbaum considers "the wrong central issue for budding AI scientists to learn".

In his proposal of *intelligence without representation*, Brooks (1991) argues for the following points in creating artificial intelligence:

- The capabilities of intelligent systems must be incrementally build.
- At each step, complete intelligent systems must be built and let loose in the real world with real sensing and real action.

From these points, he concludes that using the world as a model is better than explicit representations. This conclusion, apparently in contradiction with the declarative representation approach, is nevertheless supported by the proposed model, for the information waves used as declarative representation in the proposed model are incrementally built, and obtained directly from the environment through interaction, and they represent exactly the relevant features of the real world that count for the cognitive agent being modelled.

In his response to Brooks' paper, Kirsh (1991b) defends the necessity of concepts in the cognitive skills, and states that: "to have a concept is (...) to have the capacity to find an invariance across a range of contexts, and to reify that invariance so that it can be combined with other appropriate invariances." This statement is also supported by the proposed model, as it perfectly describes the interaction of the propagation and categorization levels of the model.

Pribram (1971), a biology researcher, proposes an "holographic metaphor" that presents many related points with the proposed model. He states that sensory input is transformed into a brain wave, and that this wave travels to an area of the brain that interprets its meaning. The interpretation is a product of various kinds of standing memory waves. These waves travel across the brain simultaneously and interfere with each other. The arguments that make Pribram's methaphor interesting to the biology community, even if he does not propose a precise mechanism to explain how the methaphor works, could be used also to defend the proposed model, with the difference that what is proposed is a formal model, where the mechanisms are known. The arguments are that the methaphor explains: (i) the distribution of memory, (ii) the associational characteristic of the memory, (iii) the immense amount of information stored in a limited volume, (iv) the mechanism of the brain as an open cybernetic system of organism plus environment.

Discussing alternative metaphors for Artificial Intelligence, West and Travis (1991) state three criteria that would argue in favor of a metaphor for Artificial Intelligence:

- Suggestiveness: the generation of referents, on both sides of the metaphoric relationship, that can be used to confirm or dispute the validity of the metaphor.
- Concreteness: the generation of practical avenues of research or testable hypotheses.
- Consistency: internally and in relation to what is known or believed about the mind and brain in other discuplines.

We would like to conclude that, not the formal model which is just an attempt to demostrate the possibility of formalizing the proposed general approach, but this general approach itself satisfies, at least partially, these three criteria.

7 Conclusion

We presented a unified framework to model the cognition that integrates symbolic reasoning, neural network theory and natural selection principles. This framework is based on the postulate that there is no discontinuity between physical, organic and cognitive mechanisms (Lupasco, 1974).

The wave propagation formalism of a single cognitive agent has been implemented using Common Lisp. This system, with its graphical interface, is being used to study the geometric representations of the states of the propagation space. The states where no further inferences are possible are of special interest. These "steady states" represent complete coherent states of knowledge that can be reached from several different sets of initial pieces of information. Geometric manipulations that transform one of these states into another suggest a type of analogy mechanism: they map a coherent knowledge state into another that is supported by different initial informational configurations.

References

1. BELNAP, N.D., *A Useful Four-Valued Logic.* In "J.M. Dunn fand G. Epstein (Editors), Modern Uses of Multiple-Valued Logics", D. Reidel Pub. Co., 1977.
2. BIRNBAUM, L., *Rigor mortis: a response to Nilsson's "Logic and Artificial Intelligence".* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 57-77, January 1991.
3. BITTENCOURT, G., *A Hybrid System Architecture and its Unified Semantics.* In "Z.W. Ras (Editor), Proceedings of The Fourth International Symposium on Methodologies for Intelligent Systems", Charlotte, North Caroline, USA, October 12–14, pp. 150-157, 1989.
4. BITTENCOURT, G. *An Architecture for Hybrid Knowledge Representation.* Ph.D. Thesis, Universität Karlsruhe, Deutschland, 31 Januar 1990.
5. BITTENCOURT, G. *Space Embodiment and Natural Selection.* Internal Report, INPE, 1993.
6. BROOKS, P.A., *Intelligence without Representation.* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 139-159, January 1991.

7. CHANGEUX, J.-P., *L'Homme Neuronal.* Collection Pluriel, Librairie ArthΩme Fayard, 1983.
8. HAMPDEN-TURNER, C., *Maps of the Mind: Charts and Concepts of the Mind and its Labyrinths.* Collter, New York, 1981.
9. HEARN, A.C., *REDUCE User's Manual: Version 3.3.* RAND Publication CP78, The RAND Corporation, Santa Barbara, CA, April 1987.
10. KIRSH, D., *Foundations of AI: the Big Issues.* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 3-30, January 1991a.
11. KIRSH, D., *Today the Earwig, Tomorrow Man ?* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 161-184, January 1991b.
12. LUPASCO, S., *L'énergie et la matière vivante. Antagonisme constructeur et logique de l'hétérogène.* Juillard, Paris, 1974.
13. McCARTHY, J. and HAYES, P.J., *Some Philosophical Problems from the Standpoint of Artificial Intelligence.* In "D. Mitchie and B. Meltzer (Editors), Machine Intelligence 4", Edimburgh University Press, Edimbourgh, GB, pp. 463-502, 1969.
14. MINSKY, M.L. and PAPERT, S.A., *Perceptrons: An Introduction to Computational Geometry.* M.I.T. Press, 1969.
15. NEWELL, A. and SIMON, H.A., *Computer Science as Empirical Inquiry: Systems and Search.* Communications of the ACM, Vol. 19, No. 3, pp. 113-126, March 1976.
16. NILSSON, N.J., *Logic and Artificial Intelligence.* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 31-56, January 1991.
17. NORMAN, D.A., *Approaches to the Study of Intelligence.* Artificial Intelligence (Special Volume Foundations of Artificial Intelligence), Vol. 47, No. 1-3, pp. 327-346, January 1991.
18. PATEL-SCHNEIDER, P.F , *A Decidable First-Order Logic for Knowledge Representation.* Proceedings of IJCAI 9, pp. 455–458, 1985.
19. PRIBRAM, K., *Language of the Brain: Experimental Paradoxes and Principles in Neuropsychology.* Englewood Cliffs, N.J., Prentice Hall, 1971.
20. RUMELHART, D.E. and McCLELLAND, J. (Editors), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1: Foundations.* M.I.T. Press, Cambridge, MA, 1986a.
21. RUMELHART, D.E. and McCLELLAND, J. (Editors), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition 2: Psychological and Biological Models.* M.I.T. Press, Cambridge, MA, 1986b.
22. SMOLENSKY, P., *Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems.* Artificial Intelligence (Special Issue on Connectionist Symbol Processing), Vol. 46, No. 1-2, pp. 159-216, January 1991.
23. TORRANCE, S. (Editor), *The Mind and the Machine: Philosophical Aspects of Artificial Intelligence.* Ellis Horwood series in Artificial Intelligence, John Wiley & Sons, 1984.
24. WEST, D.M. and TRAVIS, L.E., *From Society to Landscape: Alternative Metaphors for Artificial Inelligence.* AI Magazine, pp. 69-83, Summer 1991.
25. ZADEH, L.A., *Fuzzy Logic and Approximate Reasoning.* Synthese, Vol. 30, pp. 407-428, 1975.

Multi-Context Systems as a Tool to Model Temporal Evolution

Mauro Di Manzo and Enrico Giunchiglia

Mechanized Reasoning Group
DIST - University of Genoa
Via Opera Pia 11A, 16145 Genoa, Italy
enrico@dist.unige.it

Keywords: Context, Multi-Context Systems

Abstract. *Contexts* are defined as axiomatic formal systems. More than one context can be defined, each one modeling/solving (part of) the problem. The (global) model/solution of the problem is obtained making contexts communicate via *bridge rules*. Bridge rules and contexts are the components of Multi Context systems. In this paper we want to study the applicability of multi contexts systems to reason about temporal evolution. The basic idea is to associate a context to each temporal interval in which the "model" of the problem does not change (corresponding to a state of the system). Switch among contexts (corresponding to modifications in the model) are controlled via a meta-theoric context responsible to keep_track_of the temporal evolution. In this way (i) we keep a clear distinction between the theory describing the particular system at hand and the theory necessary for predicting the temporal evolution (ii) we have simple object level models of the system states and (iii) the theorem prover can faster analize and answer to queries about a particular state. The temporal evolution of a U-tube is taken as an exampie to show both the proposed framework and the GETFOL implementation.

1 Introduction

Recent work in artificial intelligence [GW88, Giu91, McC90, McC91, Sho91] advocates for the use of multiple formal systems (called *contexts*) both in knowledge representation and in problem solving. Even if there is not a consensus about how contexts should be (formally) defined, there is the common intuition that many (reasoning) phenomena are "better" modeled as a set of interacting contexts each one modeling only part of the whole.

This paper investigates the applicability of contexts to model the temporal evolution of dynamic systems. We define a context as an axiomatic formal system modeling *what is known or assumed* about the system in an instant/interval of time (a *state* of the situation). The system temporal evolution is modeled by a tree of (temporally related) contexts, called *state-contexts (s-contexts)*. A particular metatheoretic *Problem Solving Context (PSC)* controls the evolution of the system and keeps track of the temporal evolution. *PSC* and s-contexts are

related via *bridge rules* (*i.e.* rules enabling the derivation of one fact in a context on the basis of facts derived in other contexts) [Giu91].

Three are the main advantages of the multi-contextual approach we adopt. First of all, we keep a clear distinction between the theory describing the particular system at hand and the theory necessary for predicting the system evolution. A different system can be modeled simply by specifying its *structural description* in a "new" *s*-context. The *Problem Solving Context* is the same. Second, each *s*-context provides a simple object level description of a system state. Different facts referring to different states do not cause contradiction since they are asserted in distinct contexts. Finally, when reasoning about a particular system state we may directly consider the corresponding *s*-context. As obvious consequence, the theorem prover performances are improved since the search space (in each context) is smaller.

We first provide a theoric characterization of the general framework (section 2). In section 3 we model the behavior of a U-tube as an example to illustrate both the theory and the implementation built in the GETFOL system [GT91]. Finally, in section 4 we make some final considerations and conclude with the acknowledgements (section 5).

2 Reasoning about temporal evolution

We define a context C to be an axiomatic formal system (defined as a triple consisting of a language L, a set of axioms $\Omega \subseteq L$ and a set of inference rules Δ, *i.e.*, $C = \langle L, \Omega, \Delta \rangle$). Besides the technical/logical definition a context is intended to model what the agent knows or assumes that holds while reasoning about a situation. The temporal evolution of a given system (according to the agent mental image) is modeled by a set of contexts each one associated to a particular state. Contexts communicate via *bridge rules*: rules whose premises and consequences belongs to different contexts [Giu91].

Given a family of context $\{C_i = \langle L_i, \Omega_i, \Delta_i \rangle\}_{i \in I}$ we write $\langle A, i \rangle$ to specify, when ambiguous, that A is a wff of language L_i, (A is an L_i-wff). Adopting a suitable modification of the natural deduction formalism, notation and terminology as defined in [Pra65], we write bridge rules as:

$$\frac{\langle A_1, i_1 \rangle \quad \dots \quad \langle A_n, i_n \rangle \quad \overset{[\langle B_1, j_1 \rangle]}{\langle A_{n+1}, i_{n+1} \rangle} \quad \dots \quad \overset{[\langle B_m, j_m \rangle]}{\langle A_{n+m}, i_{n+m} \rangle}}{\langle A, i \rangle}\ \delta$$

Consistently with [Pra65] notations, δ represents a rule discharging the assumptions $\langle B_1, j_1 \rangle, \dots, \langle B_m, j_m \rangle$.

Contexts and bridge rules are the elements of *multi context systems (MC-systems)*. Within MC-systems we can define a notion of derivation spanning through multiple contexts.

Definition 1 Multi-Context System. Let I be a set of indices and $\{C_i = \langle L_i, \Omega_i, \Delta_i \rangle\}_{i \in I}$, a family of contexts. A *Multi-Context Formal System* (*MC system*) MS is a pair $\langle \{C_i\}_{i \in I}, \Delta_{MS} \rangle$ where Δ_{MS} is a set of *bridge rules*. [1]

In a MC-system, *deductions* are trees of wffs built starting from a finite number of assumptions and axioms, possibly belonging to distinct languages, and applying a finite number of inference rules. $\langle A, i \rangle$ is *derivable* from a set of wffs Γ in a MC system MS ($\Gamma \vdash_{MS} \langle A, i \rangle$) if there is a deduction with bottom wff $\langle A, i \rangle$ whose undischarged assumptions are in Γ. $\langle A, i \rangle$ is a *theorem* in MS ($\vdash_{MS} \langle A, i \rangle$) if it is derivable from the empty set.

Any deduction can be seen as composed of subdeductions in distinct contexts C_j, (obtained by repeated applications of Δ_j-rules), any two or more subdeductions being concatenated by one or more applications of bridge rules.

Even if definition 1 is more general, allowing arbitrary formal languages and deductive machinery, in this paper we concentrate on first order languages. In particular we consider an MC-system $\mathcal{MS}$ composed by a countable family of contexts (the *s*-contexts) and a *Problem Solving Context (PSC)* which axiomatizes the metatheory necessary to solve the problem. Thus, in this paper *PSC* contains the basic principles for modeling the evolution of dynamic systems. The basic idea is that, given the initial description of the system in a *s*-context, we carry on the problem solving activity in *PSC*, determining a (meta-level) description of the plausible states of the system. By *reflecting down PSC* results in *s*-contexts, we get simple object-level descriptions of such states. Note that it is not true that *all* the reasoning takes place in *PSC*. In fact, given a description of the system in a *s*-context, *PSC* is only responsible for determining which is the set of descriptions of the system which are compatible with the considered one (supposing it provides only a partial description of the system), and for determining the state transition. *PSC cannot* reason about a situation. For example, it is not possible in *PSC* to deduce that β holds in C supposing we know that α and $\alpha \rightarrow \beta$ also hold in C (*i.e.* $TH(\text{“}\alpha\text{”}, \text{“}C\text{”}), TH(\text{“}\alpha \rightarrow \beta\text{”}, \text{“}C\text{”}) \not\vdash_{PSC} TH(\text{“}\beta\text{”}, \text{“}C\text{”})$). On the other hand, we can deduce β from α and $\alpha \rightarrow \beta$ via application of modus ponens directly in the C context, and *reflect up* this result in *PSC*.

Such correspondence between *PSC* facts and *s*-contexts facts, is obtained defining $\mathcal{MS}$ such that:

- the set of natural deduction rules as described in [Pra65] belong to the set of inference rules of each context in $\mathcal{MS}$.
- *PSC* has names for each *s*-context and for any formula of *s*-contexts. More precisely, we write “w” and “C” to indicate the *PSC* names for the formula w and the context C respectively.
- in *PSC* there is a distinguished predicate $TH(\text{“}w\text{”}, \text{“}c\text{”})$ which holds iff the formula w belonging to the language of context c is a theorem (*i.e.* $\vdash \langle w, c \rangle$). This is obtained by defining the *reflection principles* [GS89]:

$$\frac{\langle w, C_i \rangle}{\langle TH(\text{“}w\text{”}, \text{“}C_i\text{”}), PSC \rangle} R_{down} \qquad \frac{\langle TH(\text{“}w\text{”}, \text{“}C_i\text{”}), PSC \rangle}{\langle w, C_i \rangle} R_{up}$$

[1] Note that for some $i, j \in I$, we may have $C_i = C_j$

to be the set of $\mathcal{MS}$ bridge rules.

So far, *PSC* represents a correct and complete metatheory of provability in *s*-context. What we want to do is to *use PSC* to predict the behavior of the system described in *s*-contexts. For such a task, it is necessary to define a a *PSC* predicate allowing us to make *"non monotonic"* extensions to the situation described in a *s*-context. We thus introduce a predicate *CBB*(w,p,c), to mean that the wff w *Can_Be_Believed* wrt the problem p in context c (notice that we are assuming to have a many-sorted language as GETFOL-language is). Notice that we get the set of facts in which we can believe, not only dependent on the situation (*i.e.* the *s*-context), but also on the *problem* we are reasoning about. In fact, considering a situation, we may have different beliefs and/or reasoning principles (leading to different set of facts that *can be believed*) depending on the problem we are facing. In any case, the wff w cannot be asserted in the context c, since it may invalidate other beliefs we may have about the situation, but need to be asserted in a "new" *s*-context. This is obtained via definition of a injective function $mkcxt(w, p, c)$. *mkcxt* is such that, applied to ground terms, gives a constant of sort context (being the *PSC* name of an *s*-context). We recall that by definition of injective function, *mkcxt* satisfies the following fact:

$$mkcxt(w_1, p_1, c_1) = mkcxt(w_2, p_2, c_2) \rightarrowtail (w_1 = w_2 \wedge p_1 = p_2 \wedge c_1 = c_2)$$

To keep tracks of the state transictions in a compact way, we introduce a predicate *Context-To-Context (C2C)*. $C2C(w, p, c_1, c_2)$ takes into account the fact that context c_2 has been introduced (in the deduction) while solving problem p in context c_1.

We are now able write the PSC axioms stating what we have informally described above:

$$PSC1 : \forall wpc.(CBB(w, p, c) \rightarrowtail TH(w, mkcxt(w, p, c)))$$
$$PSC2 : \forall wpc_1c_2.(mkcxt(w, p, c_1) = c_2 \rightarrowtail C2C(w, p, c_1, c_2))$$

Summarizing, what we have defined are some basic principles to:

- maintain a "correct" relation among *PSC* and *s*-contexts (via the *TH* predicative letter and the reflection principles);
- switch *s*-context when we want to consider some (new) situation (via predicative letters *CBB*, *TH* and axiom *PSC*1) and keep track of the context-transition (via predicative letter *C2C* and axiom *PSC*2).

What it remains to specify are the problem specific criteria by which a formula can be believed in a context wrt a problem (thus causing a state transition), and which facts holding in the context we were reasoning also hold in the newly considered one.

3 Solving the U-tube example

In this section we show how it is possible to extend the basic framework described in the previous section to model the behavior of a particular dynamic system. Two solutions are proposed for each addressed problem: a simple-mind one and a slightly more elaborated one relying on some principles commonly used in Qualitative Reasoning [Bob84, WDK89]. To give a flavour of the **GETFOL** implementation, from here on we use **GETFOL** syntax and notations (using the **typewriter** font to evidentiate **GETFOL** code and symbols). We consider a simplified version of the U-tube example [For84, Kui86]. We take into account only two quantities, being the value of the height the liquid reaches in the first and second container. We call such quantities respectively `H1` and `H2`, and define them as the only two constants in each s-context. Three different "initial states" are "possible":

```
H1 > H2          H1 = H2          H2 > H1
```

Depending on the initial state we have three different behaviors:

1. `H1 > H2`, in such a case there will be a fluid-flow from the first container to the second, until `H1=H2` ("final state").
2. `H1 = H2`, in such a case nothing happens and the system is stable.
3. `H1 < H2`, in such a case there will be a fluid-flow from the second container to the first, until `H1=H2` ("final state").

Such facts are formally stated by the following axiom:

```
DECLARE INDCONST H1 H2;
AXIOM STRCTDES:
   (H1 > H2 imp (INCREASE(H2) and DECREASE(H1))) and
   (H1 = H2 imp (CONSTANT(H1) and CONSTANT(H2))) and
   (H2 > H1 imp (INCREASE(H1) and DECREASE(H2)));
```

formalizing the *structural description* of the U-tube. We take such an axiom to be part of the set of axioms in each s-context. For lack of space, we omit the axioms about "$\geq$" (*reflexivity, antisymmetry* and *transitivity*) and define "$v_1 > v_2$" as "$(v_1 \geq v_2) \wedge \neg(v_1 = v_2)$". Such axioms should be declared in **GETFOL**. From a theoric point of view, we consider them as part of the logical axioms.

Note that in each s-context we have a *finite* number of variables and individual constants, making the theory associated to each s-context *decidable* [Kle52]. So, it makes sense asking wether one fact is consistent with the theory corresponding to an s-context. In the following, we make use of this facility.

Considering *PSC*, in *PSC* we have sorts **WFF**, **CONTEXT**, **PROBLEM** and **VALUE**. For each sort, we suppose to have denumerately many variables obtained from the first letter of the sort, eventually concatenated with a number. Moreover

- "H1" and "H2" are defined to be constants of sort VALUES and represent the *PSC*-names for the *s*-contexts constants H1 and H2 respectively;
- function symbols mkmj, mkeq maps pair of values into wffs. If "H1", "H2" are the *PSC*-names for H1, H2 respectively, mkmj("H1","H2") and mkeq("H1","H2") are the *PSC* names for H1 > H2 and H1 = H2 respectively;
- in the same way, we have function symbols mkconst, mkincr, mkdecr mapping a value into a formula. mkconst("H1"), mkincr("H1"), mkdecr("H1") represent the names for *s*-context formulas CONSTANT(H1), INCREASE(H1), DECREASE(H1) respectively.

We want to show how, supposing we do not know which of the three possible relations between H1 and H2 initially holds, it is possible

- to create the three contexts corresponding to the three possible initial states. In such a phase we say that we solve the "*complete*" problem (since we disambiguate the initial only partially known situation).
- to determine the next state of behavior, starting from each possible initial situation. In this phase, we say that we are solving the "*predict*" problem (since we predict the next state of behavior).

Different reasoning principles are to be used depending on the problem to be solved. To address separetely the two problems, we define two constants (of sort PROBLEM) in the *PSC* context:

```
DECLARE SORT PROBLEM;
DECLARE INDCONST CMPLT PRDCT [PROBLEM];
```

In the following sections we omit some of GETFOL declarations and consider only those relevant for the comprehension of the example.

3.1 Solving the complete problem

We suppose to start our reasoning in the context C0 in which the only fact we know is the structural description of the system. The simplest solution of the complete problem should be to assume in *PSC* that each relation between H1 and H2 *can_be_believed*, apply axiom *PSC*1 and derive (depending on such assumptions) each relation in three separated contexts C1, C2 and C3.

Supposing we want a theory *proving* such results, then we have to consider the following axiom:

```
AXIOM CMPLT1: forall v1 v2 c.
        (CONSIST(mkmj(v1,v2),c) imp CBB(mkmj(v1,v2),CMPLT,c)) and
        (CONSIST(mkeq(v1,v2),c) imp CBB(mkeq(v1,v2),CMPLT,c)) and
        (CONSIST(mkmj(v2,v1),c) imp CBB(mkmj(v2,v1),CMPLT,c));
```

It simply says that a relation between to values may hold in a situation, if it is consistent with the fact we know about such a situation. Supposing that the theory associated to each s-context is decidable, (as it is in our case), we may also define an inference with no premisess and conclusion `CONSIST("w","C")` with the obvious restriction that w must be consistent in C.

If s-context c_2 has been introduced while solving the complete problem in context c_1, then all the facts which are true about the state corresponding to c_1 are also true in the state described by c_2:

```
AXIOM CMPLT2: forall w c1 c2. (C2C(w,CMPLT,c1,c2) imp
                          forall w1. (TH(w1,c1) imp TH(w1,c2)));
```

In our case, supposing we are in a initial situation `C0` in which we do not know the relation holding between `H1` and `H2`, we can determine the three possible extensions as the result of this inferential activity in *PSC*:

1. Instantiate axiom `CMPLT1` with `"H1","H2","C0"`. Prove the hypotheses of the implication and prove:

   ```
   CBB(mkmj("H1","H2"),CMPLT,"C0"),
   CBB(mkeq("H1","H2"),CMPLT,"C0"),
   CBB(mkmj("H2","H1"),CMPLT,"C0").
   ```

2. Then, via axiom *PSC1* we can prove:

   ```
   TH(mkmj("H1","H2"),mkcxt(mkmj("H1","H2"),CMPLT,"C0"),
   TH(mkeq("H1","H2"),mkcxt(mkeq("H1","H2"),CMPLT,"C0"),
   TH(mkmj("H2","H1"),mkcxt(mkmj("H2","H1"),CMPLT,"C0").
   ```

3. Supposing we get

   ```
   mkcxt(mkmj("H1","H2"),CMPLT,"C0")="C1"
   mkcxt(mkeq("H1","H2"),CMPLT,"C0")="C2"
   mkcxt(mkmj("H2","H1"),CMPLT,"C0")="C3"
   ```

   by substitutions of equals, we have:

   ```
   C2C(mkmj("H1","H2"),CMPLT,"C0","C1")   TH(mkmj("H1","H2"),"C1")
   C2C(mkeq("H1","H2"),CMPLT,"C0","C2")   TH(mkeq("H1","H2"),"C2")
   C2C(mkmj("H2","H1"),CMPLT,"C0","C3")   TH(mkmj("H2","H1"),"C3")
   ```

   and, by means of the *reflection down* principle, assert that `H1` is $>, =, <$ of `H2` in `C1`, `C2`, `C3` respectively.

Note that in this case it is useless to apply axiom `CMPLT2` since all the theorems in `C0` (*i.e.* $\{w : \vdash_{MS} \langle w, C0 \rangle\}$) also hold in `C1`, `C2` and `C3`.

3.2 Solving the predict problem

Considering the situation modeled in context C1 in which H1 > H2, and supposing we want to predict the next relation plausibly holding between the two values, we could assume CBB("H1=H2",PRDCT,"C1"), and assert (via $PSC1$ and R_{down}) H1=H2 in a context C4.

A more refined solution, which obeys the *limit analysis* as usually carried out in Qualitative Reasoning [DKB84, For84, Kui86]) is also not difficult to be provided. In fact, it is sufficient that one value v_1 is greater of another v_2 and that the former is increasing or the latter decreasing, for believing that there is a possible situation in which $v_1 = v_2$ holds [Kui86]:

```
AXIOM PRDCT1:forall v1 v2 c.
((TH(mkmj(v1,v2),c) and (TH(mkincr(v2),c) or TH(mkdecr(v1),c))) imp
                                        CBB(mkeq(v1,v2),PRDCT,c));
```

In the same way, if two values are equal, it is sufficient that one value is changing for believing that they are going to be one greater than the other:

```
AXIOM PRDCT5:forall v1 v2 c.
((TH(mkeq(v1,v2),c) and (TH(mkincr(v1),c) or TH(mkdecr(v2),c))) imp
                                        CBB(mkmj(v1,v2),PRDCT,c));
```

Of course the axioms are so simple since we consider a very simple quantity space $(-, 0, +)$ for derivative values. However the framework is easily extendible to treat more complex quantity spaces.

Considering the U-tube example, by the facts we have derived solving the "complete" problem and from the general description, we have:

in C1 : H1>H2, INCREASE(H2), DECREASE(H1)
in C2 : H1=H2, STEADY
in C3 : H2>H1, INCREASE(H1), DECREASE(H2)

For example, let consider how the state in C1 evolves:

1. Prove TH(mkmj("H1","H2"),"C1") and TH(mkincr("H1"),"C1") via reflection up.
2. Instantiate axiom PRDCT1 with "H1","H2","C1" and prove:

   CBB(mkeq("H1","H2"),PRDCT,"C1");

3. suppose we get mkcxt(mkeq("H1","H2"),PRDCT,"C1")="C4", we have:

   C2C(mkeq("H1","H2"),PRDCT,"C1","C4") TH(mkeq("H1","H2"),"C4")

4. By means of the *reflection down* principle assert H1=H2 in C4.

4 Final considerations

What have we achieved?

From a representational point of view, we have a simple model of each state and in each s-context we have only those facts which are "relevant" for the state being modeled (with clear computational advantages). We have a clear distinction between the (FOL) model of the states and the metatheoretic principles allowing us to derive the evolution. In *PSC* we have a complete knowledge of the whole tree of states: this enables us to consider and apply whatever constraint we want about the evolution. For example, supposing we want to consider (metatheoretic) principles enabling us to cut off "useless" or "impossible" states (*e.g.* determined via some principles like those in [KC87]), we can simply suppose to mark such states (that is s-contexts) as "no good" (*e.g.* by having a *PSC* predicate $NOGOOD(c)$). Such s-contexts are not to be taken into account while generating the evolution tree, and this can be obtained simply putting in the hypotheses of each axiom of *PSC*, the condition that we are speaking about "good" context.

From a formal point of view, inside each context we have all the nice properties FOL has (e.g. clear semantics and proof theory). Solving the predict problem, we avoid contradiction by isolating each state in separated s-contexts: in [GW88] this kind of non-monotonicity has been defined *non essential*. Solving the complete problem, we make "consistency-based" extension to the theory. Key point in our framework is axiom $PSC1$ which allows (together with the *reflection principles*) to safely define the s-contexts corresponding to non monotonic extension of *PSC*.

We also provide an implementation within the GETFOL system [GT91]. GETFOL is an interactive system based on Natural Deduction [Pra65] extending the FOL system developed at Stanford by Richard Weyhrauch [Wey80]. GETFOL turns out to be appropriate for our purposes since, among othe features (*e.g.* the many sorted language), allows the user to define and handle multiple contexts.

5 Acknowledgements

We are indebted to Fausto Giunchiglia for the many invaluable GETFOL&FOL suggestions we had. The *Mechanized Reasoning Group* in Trento and Genova, the *Formal Reasoning Group* in Stanford, Marco Cadoli, John McCarthy, Carolyn Talcott and Richard Weyhrauch are also thanked for having helped the second author in various ways. This work has been supported by the Italian National

Research Council (CNR), Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo (Special Project on Information Systems and Parallel Computing).

References

[Bob84] D.G. Bobrow. Qualitative reasoning about physical systems. *Artificial Intelligence.*, 24, 1984.

[DKB84] J.H. De Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence.*, 24:7–83, 1984.

[For84] K.D. Forbus. Qualitative process theory. *Artificial Intelligence.*, 24:85–168, 1984.

[Giu91] F. Giunchiglia. Multilanguage systems. In *Proceedings of AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, 1991.* Also IRST-Technical Report no. 9011-17.

[GS89] F. Giunchiglia and A. Smaill. Reflection in constructive and non-constructive automated reasoning. In H. Abramson and M. H. Rogers, editors, *Proc. Workshop on Meta-Programming in Logic Programming*, pages 123–145. MIT Press, 1989. Also IRST-Technical Report 8902-04 and DAI Research Paper 375, University of Edinburgh.

[GT91] F. Giunchiglia and P. Traverso. GETFOL User Manual - GETFOL version 1. Manual 9109-09, IRST, Trento, Italy, 1991. Also MRG-DIST Technical Report 9107-01, DIST, University of Genova.

[GW88] F. Giunchiglia and R.W. Weyhrauch. A multi-context monotonic axiomatization of inessential non-monotonicity. In D. Nardi and P. Maes, editors, *Meta-level architectures and Reflection*, pages 271–285. North Holland, 1988. Also MRG-DIST Technical Report 9105-02, DIST, University of Genova, Italy.

[KC87] B.J. Kuipers and C. Chiu. Taming intractable branching in qualitative simulation. In *Proc. IJCAI conference*, pages 1079–1085. International Joint Conference on Artificial Intelligence, 1987.

[Kle52] S.C. Kleene. *Introduction to Metamathematics.* North Holland, 1952.

[Kui86] B.J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[McC90] J. McCarthy. Generality in Artificial Intelligence. In J. McCarthy, editor, *Formalizing Common Sense - Papers by John McCarthy*, pages 226–236. Ablex Publishing Corporation, 1990.

[McC91] J. McCarthy. Notes on formalizing context. Unpublished, 1991.

[Pra65] D. Prawitz. *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[Sho91] Y. Shoham. Varietes of context. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation - Papers in honor of John McCarthy*, pages 393–408. Academic Press, 1991.

[WDK89] D.S. Weld and J.H. De Kleer. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann Publishers, Inc., 95, First Street, Los Altos, CA 94022, 1989.

[Wey80] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artif. Intel.*, 13(1):133-176, 1980.

Systematic assessment of temporal reasoning methods for use in autonomous agents

Erik Sandewall
Department of Computer and Information Science
Linköping University
Linköping, Sweden
E-mail ejs@ida.liu.se

Abstract. Non-monotonic logics for reasoning about time and action have traditionally been verified through their informal plausibility and their application to a small number of test examples. I present a systematic method using an *underlying semantics*, where a preferential non-monotonic logic is correct if the maximally preferred models equals the set of histories that can arise in a "scenario simulation" of the underlying semantics. Several old and new logics have been analyzed using the systematic method.
The systematic method has first been applied to the relatively straightforward case of scenarios with simple inertia, where non-deterministic actions and actions with duration different from one timestep are the only complications. In this extended abstract I first make a brief review of the approach to simple inertia, and then describe how the systematic approach can be extended to obtain a precise account of broader types of frame problems, such as problems involving surprises and ramifications.

1 The systematic approach

There has been much research in recent years on methods for reasoning about actions and change, and on finding solutions for the so-called "frame problems". The non-monotonic logics which have been proposed for this purpose have traditionally been verified through their informal plausibility and their application to a small number of test examples. The limitations of this approach have become more apparent in recent years, and at the same time some results of more systematic nature have been presented in particular by Lin and Shoham[LS91], Lifschitz[Lif91], and Reiter[Rei91], all of which have used a situation-calculus framework. In this extended abstract I shall describe another approach which is based on an explicit treatment of time in a linear time domain, and which is able to accomodate a broader range of reasoning problems. The complete results are available in a review version of a forthcoming book[San92].

By an *assessment* of a nonmonotonic logic, I mean a statement that for a certain class of reasoning problems, the logic is guaranteed to always give the correct set of conclusions: the set of intended conclusions is equal to the

set of actual conclusions obtained through the logic. A *systematic approach* to nonmonotonic logic is one where it possible to obtain such assessments with formal proofs, based on precise definitions. A systematic approach requires in particular that one makes precise definitions of the class of reasoning problems in question, and of the set of intended conclusions.

I have developed such a systematic approach in model theoretic terms, where therefore the correctness criterion is that the set of intended models shall equal the set of actually obtained models in the logic. A central part of the approach is to use an *ontological taxonomy*, which provides the instrument for characterizing classes of systems or "worlds" which the logic is used for reasoning about. For a given logic, the assessment will specify a class within that taxonomy for which the logic always obtains the correct results. This makes it easy to compare the range of applicability of different logics.

The nonmonotonic logics that are to be assessed and compared in this fashion, can often be characterized in terms of a *preference relation* or a *selection function* on models. A selection function maps any set S of interpretations to a subset of S, and is used to map from the set of classical models for the given premises to a smaller set of selected models. In one important special case, the selection function obtains those members of S which are minimal according to a certain preference relation, so the maximally preferred models are the selected ones. More complex selection functions are needed to account for broader classes of reasoning problems. In particular one may need to divide the premises into several *partitions*, and to apply different selection functions to the classical model sets for the different partitions.

The main steps in the development of the approach are as follows.

1. Semiformal definition of an *overview taxonomy*, using a set of *specialities* i.e. features of reasoning problems. The following are some of the ontological specialities, i.e. those which characterize the world that is being reasoned about:

   C: Concurrent actions are allowed.

   L: Delayed effects are allowed, i.e. effects of an action that take place after the action has been completed.

   T: Timebound objects are allowed, i.e. objects that are created and terminated at some points in time.

   The overview taxonomy also specifies the epistemological assumptions, i.e. what assumptions are made about the completeness of the information in the given premises. Here I shall only consider the case that all actions are known, and that all potential effects of actions are also known although nondeterministic actions are allowed. This case will be represented as $\mathcal{K}$.

   The simplest category in the overview taxonomy is $\mathcal{K}$**−IA** representing the $\mathcal{K}$ assumption, integer time, sequential actions, an inertia assumption for all fluents i.e. the value of a fluent does not change except directly due to an action, but non-deterministic actions and actions with a duration in time (not just a single time-step) are allowed. Broader categories are formed e.g. as $\mathcal{K}$**−IAC** if the requirement for sequential actions is dropped, or $\mathcal{K}$**−IAL**

if delayed effects are allowed.

2. For each category of reasoning problems in the overview taxonomy, one defines a corresponding *underlying semantics* in a formal and precise fashion. I use an underlying semantics which captures the basic A.I. intuitions, similar to the "agent model" of Genesereth and Nilsson [GN87]. The primary purpose of the underlying semantics is to define what is the set of intended models for a given set of premises. Indirectly, the underlying semantics also serves as a precise definition of the problem family at hand. Also it is the basis for the subsequent steps of analysis.
3. Next, a more fine-grained taxonomy is defined within the overview category at hand, by introducing subordinate specialities. For example, within the $\mathcal{K}-\mathbf{IA}$ category we define the more restricted classes $\mathcal{K}-\mathbf{IsA}$ where all actions must be performed over a single time-step, $\mathcal{K}-\mathbf{IAd}$ where all actions must be deterministic (the state of the world when the action starts determines completely the state of the world when the action ends), and a number of other classes. These sub-specialities can also be freely combined, e.g. as $\mathcal{K}-\mathbf{IsAd}$.
4. Naturally there also needs to be a certain number of syntactic definitions in order to specify how formulae are to be written and organized. Besides the usual kind of syntax definitions that are needed in any logic, some of the non-monotonic logics require that one separates the given premises into several distinct sets called *partitions* – for example one set of "action statements" and another set of "observations" – and there may by special constraints on the syntax of formulae in each of those premise partitions. The syntax in this extended sense must be related to the underlying semantics as well as to the conventional (Tarskian) semantics for the logic.
5. At this point all the conceptual tools are in order, and the assessment can be performed. The result of the assessment must be a statement that for a given class of reasoning problems which is defined in the fine-grained taxonomy and for a given non-monotonic logic, the set of intended models as defined by the underlying semantics equals the set of selected models as obtained by the logic. For example if the selection function in the non-monotonic logic is based on a preference relation $\ll$, then it is the set of $\ll$-minimal models that must equal the set of intended models. Naturally a formal proof of the assessment statement is also required.

This methodology de-dramatizes the issue of correctness for an applied non-monotonic logic. If the range of applicability for logic A is a subset of the range for the logic B, then there may still be good reasons for using logic A, namely if one's application satisfies the restrictions that A imposes and if A can be implemented more efficiently than B. This is particularly important since logics (or more precisely, non-monotonic entailment criteria) that suffice for a broad range of common-sense reasoning problems tend to become quite complicated.

This systematic methodology was first applied to the case of simple inertia, where non-deterministic actions and actions with duration different from one timestep are the only complications. The results for simple inertia have been

presented in the book manuscript[San92] and in a separate paper [San93]. In this extended abstract I will describe how the results for simple inertia can be extended to obtain a precise account of broader types of frame problems, such as problems involving surprises and ramifications. However I must begin with a brief review of the approach to simple inertia since the new results build on it.

In order to keep this paper short and to focus on the general approach and the main results, I omit a number of details and fine points in the formal definitions. Please refer to the book manuscript if a clarification is needed.

2 Simple inertia

It is natural to first address the simple category of $\mathcal{K}-\mathbf{IA}$, and to analyze it in detail. This family of reasoning problems is broader than the ones which have been considered by Shoham, Lifschitz, and Reiter in particular by allowing the combination of actions with extended duration in time and non-deterministic actions such as e.g. flipping a coin. For non-deterministic actions the assumption of complete knowledge is that there are several possible outcomes of the action, even for a given state of the world when the actions starts, but the *set of possible outcomes* is known. More generally, to also account for actions with extended duration, the *set of possible trajectories* is assumed as known. Each trajectory specifies the duration of the action, the set of fluents which are changed in the course of the action, and the value of each of those fluents in each time-step within the trajectory. In other words each trajectory is a finite sequence of partial states of the world. Corresponding to the frame assumption of inertia, all fluents which are not included in the trajectory are assumed to remain constant for the duration of the action.

The underlying semantics for $\mathcal{K}-\mathbf{IA}$ captures these concepts, and is defined as follows. If r is a state of the world and A is an action, then $\mathtt{Infl}(A, r)$ is a set of features (names of fluents), and $\mathtt{Trajs}(A, r)$ is a set of trajectories each of which is a finite sequence of partial states where exactly the features in $\mathtt{Infl}(A, r)$ are assigned values. This means that the set of influenced fluents must be kept constant throughout a trajectory, and must be the same for all trajectories for a given A and r.

A reasoning problem is assumed to be stated in terms of (1) a set of *action laws* characterizing the effects of actions, (2) a set of *action statements* specifying which actions are performed and restrictions on their timing, and (3) a set of *observations* specifying the values of fluents at one or more points in time. For a given set of action laws, one can construct the corresponding functions `Infl` and `Trajs` with the structure which has just been defined, and consider them as the operative definition of the "world". The full definitions of the underlying semantics are such that the actions laws will uniquely define the world $\langle \mathtt{Infl}, \mathtt{Trajs} \rangle$.

The set of intended models for a given set of premises is then defined as follows. Consider a non-deterministic game between an "ego" and the world, where the two players take turns to perform moves, and a history in the world

is constructed as the game proceeds. The ego is essentially Newell's "knowledge level", and the "world" is understood as the combination of the agent's physical environment and its own physical body or vehicle.

At the start of the game time has the value zero, and the world is in an arbitrary state. At each step in the game, one has constructed a history over a finite interval of time $[0, n]$ where n represents "now". Each move by the ego consists of issuing an internal command to perform an action A at time n. The only effect of that invocation in the history is to add the information that action A started at time n and is still in process.

When the world has the move, it implements the action that the ego initiated, as follows. Let r be the state of the world at time n i.e. the present ending state of the history, and suppose the ego has invoked action A starting at time n. The world selects an arbitrary member of $\mathtt{Trajs}(A, r)$. Suppose that member has length k. The world will then extend the present history to length $[0, n + k]$ by adding k new states which are obtained from the chosen trajectory by allowing all other features (except those included in $\mathtt{Infl}(A, r)$) to remain constant. Then it is the ego's turn to move again. When the ego has run out of moves, the constructed finite history is extrapolated to infinity without any changes of state.

For any given reasoning problem, one uses its action laws to construct $\langle \mathtt{Infl}, \mathtt{Trajs} \rangle$, and then constructs the set of all histories (games) for all possible initial states, all possible ego behaviors and all possible choices by the world when it selects a member of $\mathtt{Trajs}(A, r)$. Among those histories, one selects those which (1) satisfy all the action statements, (2) satisfy all the observations, and (3) do not contain any actions besides those included among the action statements. The resulting set of histories is by definition the intended ones, and sets the standard for what models the non-monotonic logic is supposed to prefer or select.

With these definitions it has been possible to assess the range of applicability of a number of previously proposed entailment methods, such as different variants of chronological minimization, uses of filtering and occlusion, causal minimization, explanation closure, etc. These assessments have been presented in [San92, San93].

Notice that under this definition of intended models, it is quite possible to have inconsistent reasoning problems, where the set of intended models is empty. For example in the stolen car scenario, the assumptions are that if the car is left overnight in the garage then it is still there in the morning, and that the car is left in the garage over two successive nights. Anyway after two nights it is gone. This is an inconsistent reasoning problem in $\mathcal{K}-\mathbf{IA}$ i.e. under the underlying semantics that has just been defined. Any non-monotonic logic that obtains a non-empty set of preferred models for this example is therefore incorrect for $\mathcal{K}-\mathbf{IA}$. If one intends that the set of models in this example shall be non-empty, then one needs to use another and broader category in the taxonomy of reasoning problems. In particular the speciality **S** for "surprises" allows for additional changes of fluent values, besides those specified by $\langle \mathtt{Infl}, \mathtt{Trajs} \rangle$, which usually do not occur but which may need to be assumed in order to avoid obtaining an empty model set.

The formal analysis which leads up to assessment results for $\mathcal{K}-\mathbf{IA}$ is based

on the use of a *full trajectory normal form*, FTNF, for action laws. Consider for example a shooting world with three propositional fluents, namely a ("the turkey is alive"), l ("the gun is loaded"), and r ("it is raining"). Also there is one action *Fire* which has the effect of unloading the gun, if it was loaded, and of killing the turkey iff the gun was loaded. The rain does not affect these outcomes, and the rain itself is not affected. Furthermore for the sake of the example, assume that in the case where the turkey is alive and the gun is loaded the action takes two time units, where the gun becomes unloaded during the first time-step, and the turkey dies in the second time-step. In all other cases it takes one time-step.

Under these assumptions the set $\mathtt{Trajs}(Fire, r)$ will have exactly one member for each choice of r. Table 1 specifies, for each choice of r, the value of $\mathtt{Infl}(Fire, r)$ and the single member of $\mathtt{Trajs}(Fire, r)$. The FTNF of the action law for *Fire* is

r	$\mathtt{Infl}(Fire, r)$	$\mathtt{Trajs}(Fire, r)$
$\{a : \mathrm{T}, l : \mathrm{T}, r : \mathrm{T}\}$	$\{a, l\}$	$\langle\{a : \mathrm{T}, l : \mathrm{F}\}, \{a : \mathrm{F}, l : \mathrm{F}\}\rangle$
$\{a : \mathrm{T}, l : \mathrm{T}, r : \mathrm{F}\}$	$\{a, l\}$	$\langle\{a : \mathrm{T}, l : \mathrm{F}\}, \{a : \mathrm{F}, l : \mathrm{F}\}\rangle$
$\{a : \mathrm{T}, l : \mathrm{F}, r : \mathrm{T}\}$	$\varnothing$	$\langle\varnothing\rangle$
$\{a : \mathrm{T}, l : \mathrm{F}, r : \mathrm{F}\}$	$\varnothing$	$\langle\varnothing\rangle$
$\{a : \mathrm{F}, l : \mathrm{T}, r : \mathrm{T}\}$	$\{l\}$	$\langle\{l : \mathrm{F}\}\rangle$
$\{a : \mathrm{F}, l : \mathrm{T}, r : \mathrm{F}\}$	$\{l\}$	$\langle\{l : \mathrm{F}\}\rangle$
$\{a : \mathrm{F}, l : \mathrm{F}, r : \mathrm{T}\}$	$\varnothing$	$\langle\varnothing\rangle$
$\{a : \mathrm{F}, l : \mathrm{F}, r : \mathrm{F}\}$	$\varnothing$	$\langle\varnothing\rangle$

Table 1. The world description $\langle\mathtt{Infl}, \mathtt{Trajs}\rangle$

$$
\begin{aligned}
&[s,t]\,Load \Rrightarrow \\
&\quad ((\,[s]a \wedge l \wedge r) \Rightarrow ([s,t]a := \mathrm{F} \wedge l := \mathrm{F}) \wedge t = s+2 \wedge [s+1]a \wedge \neg l) \wedge \\
&\quad ((\,[s]a \wedge l \wedge \neg r) \Rightarrow ([s,t]a := \mathrm{F} \wedge l := \mathrm{F}) \wedge t = s+2 \wedge [s+1]a \wedge \neg l) \wedge \\
&\quad ((\,[s]a \wedge \neg l \wedge r) \Rightarrow t = s+1) \wedge \\
&\quad ((\,[s]a \wedge \neg l \wedge \neg r) \Rightarrow t = s+1) \wedge \\
&\quad ((\,[s]\neg a \wedge l \wedge r) \Rightarrow ([s,t]l := \mathrm{F}) \wedge t = s+1) \wedge \\
&\quad ((\,[s]\neg a \wedge l \wedge \neg r) \Rightarrow ([s,t]l := \mathrm{F}) \wedge t = s+1) \wedge \\
&\quad ((\,[s]\neg a \wedge \neg l \wedge r) \Rightarrow t = s+1) \wedge \\
&\quad ((\,[s]\neg a \wedge \neg l \wedge \neg r) \Rightarrow t = s+1)
\end{aligned}
$$

The construct $[s,t]f := x$ designates that the fluent f changes its value some time(s) during the time interval $[s,t]$ and finally obtains the value x. The operator $\Rrightarrow$ may be read as implication for the present purpose.

It is seen that this formula is a direct translation of the information in the table into logical formulae. The similarity is an advantage in the formal proofs, but for practical purposes one would of course use a simpler formula which is logically equivalent to the full trajectory normal form, such as

$$[s,t]\,Load \Rrightarrow$$

$$((\![s]\!]a \wedge l) \Rightarrow ([\![s,t]\!]a := \mathrm{F}) \wedge t = s+2 \wedge [\![s+1]\!]a \wedge \neg l) \wedge$$
$$((\![s]\!]\neg a \vee \neg l) \Rightarrow t = s+1) \wedge$$
$$[\![s,t]\!]l := \mathrm{F}$$

Since most of the entailment methods are defined in terms of model sets, the transformation to a classically equivalent formula does not influence their sets of selected or preferred models. However the formal proofs of assessments for various entailment methods rely on the direct correspondence between the FTNF syntax and the underlying semantics.

3 The frame problems

3.1 Towards common-sense formulations of action laws

The primary formulation of the action laws which was used in the analysis for $\mathcal{K}-\mathbf{IA}$ - the full trajectory normal form - reflects the structure of the underlying semantics. Another possible choice of primary formulation would be to use the "most natural" expression of common-sense facts in logic. As the example showed, there is a fairly big difference between these two possible choices of primary formulations of action laws due to the precise structure of the underlying semantics.

Let me refer to these choices as the *semantics-motivated* and the *common-sense-motivated* forms for expressing action laws. From an A.I. point of view it is the common-sense-motivated form(s) that one really wants to capture. However that form is notoriously difficult to pinpoint, and the semantics-motivated form has the advantage of an exact definition which makes a precise analysis possible. My approach is therefore to start with the semantics-motivated form of the action laws, and to introduce and analyze various transformations and extensions of the semantics-motivated form which may bring it closer to the common-sense-motivated forms.

This approach differs from the standard approach in the A.I. study of common-sense reasoning, which has been to start from the common-sense-motivated form of the action law, and to inquire how the inference machinery of the logic would have to be chosen in order to obtain the intended conclusions from those formulae. The problem with the standard approach is that it does not provide a reliable basis for the continued formal analysis, and useful general results have therefore not been obtained. Therefore I start instead from the underlying trajectory semantics, and ask how the logic (syntax and entailment conditions) *can be modified* in the direction of more natural or common-sense-like formulations of the action laws and of the other components of the chronicle.

A small step in this direction was already taken above with the observation that one can replace each action law in FTNF by a simplified formula which is more compact but logically equivalent (in the sense of classical logic, i.e. having the same model set). However such local rewriting will only obtain a moderate improvement of the naturalness in the action laws.

A major desideratum for more natural action laws is that one should be able to "factor out" some information from the laws i.e. to make separate statements of some of the relevant information. The action law can then be restricted to expressing the "typical" or "primary" effects of the action, whereas statements about "unusual" or "indirect" effects are written in other formulae. It seems plausible that this resembles how we tend to describe the effects of actions in common-sense terms. From an operational point of view there are several advantages with such a reformulation: the action laws will be more easily legible, and circumstances which are shared between several action laws can be stated just once rather than repeatedly inside each law. I shall use the terms *main action law* and *supplementary laws* for the two types of statements.

The main action law will be assumed to have the same syntactical form as was exemplified for plain action laws above. In particular it defines a certain number of features which are affected by the action, and (constraints on) their values during and at the end of the action. On this assumption the supplementary laws serve to characterize three phenomena:

- *Surprises:* Unusual changes in additional features besides those stated in the main law.
- *Ramification:* Normal changes in additional features, for example due to cause-effect chains.
- *Qualification:* Unusual exceptions from the main law, whereby some of the features change in another fashion (including the case of no change at all) than what is specified in the main law.

Clearly there is a certain arbitrariness as to the separation between main law and supplementary laws, and these three categories are only well defined for a given choice of main laws.

I distinguish, therefore, between the basic issue of inertia whose analysis was outlined in the previous section, and the three modularization-oriented issues of surprises, ramification, and qualification. These are essentially the same as the traditionally recognized aspects of the "frame problem". My claim here is that *the traditional frame problem can be analyzed rigorously in terms of simple inertia* (the $\mathcal{K}$–**IA** class of reasoning problems, as defined in the previous section) *plus the decomposition of action laws into main action laws and supplementary laws.*

The division that is used here differs somewhat from the usual one in the literature, since surprises are usually considered as an aspect of the "inertia problem" or of the "ramification problem", and have not been identified as a separate issue in its own right. Apart from that, the distinction between the problems of inertia, ramification, and qualification is the generally accepted one.

3.2 The systematic perspective

In order to analyze the distinction between main action laws and supplementary laws, it is not sufficient to rewrite each action law to a classically equivalent formula. Two other and stronger methods will be used:

- *Premise transformation:* Transformations on the whole chronicle, i.e. on the sets of premises including transfer of information between the premise partitions, in ways which do not change the set of selected models.
- *Extensions of the semantics:* Modifications of the underlying semantics, and corresponding addition of more premise partitions as well as definition of entailment criteria that use the additional partitions.

The shift towards common-sense-motivated forms for action laws is performed in the first case by rewriting the specific set of premises to an equivalent form, and in the second case by changing the logic itself.

Surprises and qualifications involve notions of genuine preferences and defaults: in a choice between a model that involves a surprise and one that does not, one should prefer the latter. They can therefore not be accomodated in the trajectory semantics for $\mathcal{K}-\mathbf{IA}$, so an extension of the semantics as well as additional specialities are needed. I will identify the phenomenon of surprises with the speciality code **S**, and analyze qualifications in terms of a concept of normalcy which has the speciality code **N**.

Ramifications offer a more complex picture. One possible viewpoint is to exclude ramifications from the underlying semantics. The trajectory-semantics definition of the world is retained like for $\mathcal{K}-\mathbf{IA}$, where it is assumed that the trajectories for a given action and starting state characterize *all* changes of feature values. The ramification problem is then only an issue about how a given world description, in trajectory semantics, can be correctly expressed using a combination of main action laws and supplementary laws. In such a *semantics-preserving* approach to ramification, premise transformations are the natural instrument.

The other possible viewpoint is that the world description of the form $\langle$`Infl`, `Trajs`$\rangle$ that is used for $\mathcal{K}-\mathbf{IA}$ shall only correspond to the main action laws, and that some other device is needed in the underlying semantics corresponding to the supplementary laws. This means in particular that the definition of the "game" between ego and world becomes more complex. The world's moves can no longer be described just in terms of selecting and imposing one trajectory; there must also be some *update policy* about how to modify the finite history with secondary changes corresponding to the supplementary laws. This *semantics-extending* view of ramification opens a number of possibilities, and several additional specialities are required to account for ramification in this view.

3.3 Recycling vs. embedding of entailment criteria

Since an extension of the underlying semantics will be needed for surprises, qualification, and some approaches to ramification, we shall repeatedly encounter situations where a number of entailment criteria have been defined and analyzed for a family $\mathcal{Z}$ of reasoning problems, and similar criteria are needed for a larger family $\mathcal{Z}' \supseteq \mathcal{Z}$. For example, we may wish to use our entailment criteria for $\mathcal{K}-\mathbf{IA}$ as a basis for criteria for $\mathcal{K}-\mathbf{IAS}$. There are two possible ways of proceeding:

- To check whether some of the known methods for $\mathcal{Z}$ will apply also to the extended family, or at least to some part of it which can be defined using a sub-speciality. For example, maybe some method for $\mathcal{K}-\mathbf{IA}$ will also work for a family $\mathcal{K}-\mathbf{IAS}x$ for some sub-speciality x under the speciality **S**. The existing criterion is *recycled*[1] to apply for the additional speciality.
- To define a more complex entailment criterion, for example by introducing an additional premise category or an additional preference relation. The expression for the selected models according to the new criterion contains the model-selection expression for the old criterion as a sub-expression. The new criterion is formed as an *embedding* of the existing one.

As a rule of thumb, the use of recycling ought to be preferred since it avoids the need for introducing more complexity in entailment criteria. However it is not always possible to proceed by recycling, so embedding may sometimes be the only possibility besides inventing an entirely new entailment criterion. If and when entailment methods are chosen on the basis of representative examples, one must be particularly careful not to choose a recycled criterion merely because it works for a few examples in the extended family.

4 Brief overview of assessments for surprises, ramification, and qualification

4.1 Surprises

It is clear that surprises should be minimized non-chronologically, and that it is the cardinality or the combined weight of the surprises that is to be minimized rather than the set of surprises. The only exception is that if one can assume that there is at most one surprise, then of course minimization of cardinality and of set extension is equivalent.

Since the minimization is to be done non-chronologically, then the recycling of approaches for $\mathcal{K}-\mathbf{IA}$ to apply for $\mathcal{K}-\mathbf{IAS}$ will only work for non-chronological minimization methods. However such methods have an extremely limited range of applicability. In practice it is therefore only of interest to use embedding approaches, where one makes a separate minimization of the surprises included in the models, thereby increasing by one the number of selection functions that occur in the entailment criterion.

4.2 Ramification

The "ramification problem" has been defined from the point of view of common-sense-motivated formulations for action laws, and concerns how one shall take care of all the normal, indirect effects of an action, where the *direct* effects of the action are by definition those that are expressed using the main action law. I will recognize three types of phenomena within this general definition:

[1] The term "extended" would also have been appropriate, but is already used for other purposes in non-monotonic logics.

- Changes which occur after the termination of the action, as a result of cause-effect chains with delays. This case will be counted to the **L** speciality for delayed effects, and is not being considered in this paper.
- Changes in features of objects which are not included as arguments of the action. This phenomenon will be referred to as *dependencies* or *structural ramification*. The full definition of the semantics for $\mathcal{K}-$**IA** makes an explicit assumption that there are no dependencies, and leaves the case with dependencies to the broader family $\mathcal{K}-$**IAD** of reasoning problems. Dependencies will not be further discussed here.
- Changes in additional features of objects which are included as arguments of the action, including changes in features without arguments. This will be considered as the speciality of *local ramification*, named by the speciality code **U**, and is the topic of the present subsection.

The concrete treatment of ramification in the A.I. literature is restricted to the case of local ramification. The distinction between structural and local ramification is quite important in the underlying semantics, since the proofs for assessment of currently used entailment criteria depend critically on the assumption that there are no dependencies.

One possible way of motivating ramification and of formulating supplementary laws is in terms of *constraints* i.e. propositions which are supposed to hold at all times. Constraints may have the form

$$\Box\kappa$$

where $\Box$ represents "always" and κ is a fluent formula with certain syntactic restrictions, expressing a condition that must hold momentarily at each timepoint and which does not refer to previous timepoints or to current change. Constraints of that form will be said to be *synchronic*. In the existing literature, most authors identify ramification with the use of synchronic constraints.

Semantics-preserving approaches accomodate constraints on the syntactic level, but without changing the underlying semantics or the entailment criteria. Then the constraints do not define any additional ontological speciality. Let us consider this possibility in the basic case of the $\mathcal{K}-$**IA** ontological family, with trajectory semantics. The underlying semantics defines the world in terms of a pair $\langle \mathtt{Infl}, \mathtt{Trajs} \rangle$. If the underlying semantics is unchanged, the trajectories in $\mathtt{Trajs}(A, r)$ still represent *all* changes of feature values when an action A is performed from the starting state r. Even within this rigid framework, there are two reasonable uses for constraints. *Active constraints* represent an approach to ramification; *passive constraints* actually have nothing to do with ramification and are instead a kind of derived invariants.

In the context of ramification, the purpose of constraint formulae which are used for active constraints is as an economy measure when writing out action laws. Suppose one is given a $\mathcal{K}-$**IA** world where the discrete state domain $\mathcal{R}_I$ contains only a subset of all the feature assignments that are allowed by the logic. For example, if a represents "it is dead" and d represents "it is alive", then a feature assignment where both a and d are assigned the value **T** is not

included in the set $\mathcal{R}_I$ - it is not considered to ever occur. The constraint $\Box\kappa$ characterizes this state of affairs precisely if $\mathcal{R}_I$ equals the model set for κ.

In such a case, all trajectories in $\mathtt{Trajs}(A, r)$ will be consistent with the constraints. The full trajectory normal form for the world has been assumed to enumerate all changes explicitly, but one can also consider an equivalent set of premises consisting of maximally simple action laws, which are intended to be complemented by the separate statement of the constraints. In each action law the consequent will just consist of the appropriate occlusion statements[2] for all f in $\mathtt{Infl}(A, r)$, combined with the minimal amount of other information about the action's consequences. The additional constraints on the values of the occluded (= possibly changing) fluents is provided by the supplementary laws which are constraint formulae.

Semantics-extending approaches may deal with constraints in a manner similar to surprises. The basic logic for inertia, i.e. for the ontological family $\mathcal{K}$**−IA**, defines in a very restrictive way what changes are possible, but if these definitions are applied without exception then one obtains a contradiction. In the case of surprises it was the observations that generated a contradiction and a need for escape; in the case of ramification it is instead the constraints that create this need.

Using again the "dead xor alive" constraint

$\Box(a \Leftrightarrow \neg d)$

for an example, suppose one has also the observation

$[10]a \wedge \neg d$,

saying that a is true and d is false at time 10, as well as the action statement

$[10, 12]Kill$,

saying that the "Kill" action takes place over the interval from time 10 to time 12, and the action law

$[s, t]Kill \Rrightarrow [s, t]a := \mathbf{F}$

which does not imply any possibility of change in d. Under the assumptions of $\mathcal{K}$**−IA** one will obtain an empty intended-model set, and an additional device in the underlying semantics is required in order to save the situation.

The presently most popular choice of that additional device is to use "minimization of change" or more precisely the minimization of the set of changes. However a precise analysis shows that this update policy has serious flaws which seem impossible to repair except by very drastic changes to the underlying logic.

4.3 Normality

The speciality of normality concerns what is otherwise known as the qualification problem, i.e. the cases where some of the changes that are stated in the main action law do not take place or take place differently than stated there. The current A.I. literature only contains one concrete reasoning example of qualification,

[2] Statements about the possibility of change, whose only purpose is to override the default of inertia.

namely the "potato in the tailpipe" example, and surprisingly few proposals for how to deal with the problem. Also it is quite easy to find counterexamples to those approaches, such as the "tailpipe marauder" example.

At this point I can not report any positive results about assessed entailment methods for the speciality of normality. However the underlying semantics and the other aspects of the systematic methodology has at least made it possible to define the qualification problem in more precise terms than before.

5 Conclusion

For many years, the research on common sense reasoning has been troubled by the choice between two evils. A lot of the reported research has been formal, precise, but apparently of little relevance to actual common-sense reasoning problems. A lot of other research has addressed actual problems, but using a very ad hoc methodology in particular by relying too much on representative examples of common-sense reasoning.

With the systematic methodology which is now emerging for this research, it becomes possible at last to make precise and formally proven statements about whole classes of problems and about the general applicability of logic variants, and not merely about single instances of reasoning problems. As one would expect in a formal approach to any problem area, the systematic methodology starts by analyzing a relatively simple case in depth, namely the $\mathcal{K}-\mathbf{IA}$ class of reasoning problems, and then builds on it in order to analyze successively broader and more useful classes of problems. In this way we can gradually address the plethora of logical complexities which have been assembled under the term "the frame problem", but with a firm and reliable basis at each step in the analysis.

References

[GN87] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan-Kaufmann Publishing Co., 1987.

[Lif91] Vladimir Lifschitz. Toward a metatheory of action. In *International Conf. on Knowledge Representation and Reasoning*, pages 376–386, 1991.

[LS91] Fangzhen Lin and Yoav Shoham. Provably correct theories of action (preliminary report). In *National (U.S.) Conference on Artificial Intelligence*, pages 349–354, 1991.

[Rei91] Ray Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[San92] Erik Sandewall. Features and fluents. Review version of 1992. Technical Report LiTH-IDA-R-92-30, Linköping University, Department of Computer and Information Science, 1992.

[San93] Erik Sandewall. The range of applicability of nonmonotonic logics for the inertia problem. In *International Joint Conference on Artificial Intelligence*, 1993.

GGD: Graph Grammar Developer for features in CAD/CAM

Christoph Klauck and Johannes Schwagereit

German Research Center for Artificial Intelligence Inc. (DFKI), ARC-TEC Project
Mailing address: P.O. Box 2080, D-6750 Kaiserslautern
Telephone: ++49-631-205-3477, *E-mail:* klauck@dfki.uni-kl.de

Abstract. To integrate CA*-systems with other applications in the world of CIM, one principal approach currently under development is based on feature representation. It enables any CIM component to recognize the higher-level entities – the so-called *features* – out of a lower-data exchange format, which might be the internal representation of a CAD system as well as some standard data exchange format. In this paper we present a 'made-to-measure' editor for representing features in the higher-level domain specific representation language FEAT-REP – a representation language based on a (feature-) specific attributed node labeled graph grammar. This intelligent tool, shortly called GGD, supports the knowledge engineer during the representation process by structuring the knowledge base using a conceptual language and by verifying several characteristics of the features.

1 Motivation

Research in feature-based CA*-systems like Computer Aided Design (CAD), or Computer Aided Process Planning (CAPP), has been motivated by the understanding that geometric models represent a workpiece in greater detail than it can be utilized e.g. by a designer or process planner. When CA*-experts look at a workpiece, they perceive it in terms of their own expertise – the so-called *features*. Features are domain- and company-specific description elements based on the geometrical and technological data of a workpiece that an expert in a domain associates with certain informations [2]. They are build upon a *syntax* (shape description: geometry and technology, given here by productions of a graph grammar) and a *semantics* (description of related informations, e.g. skeletal plans in manufacturing or functional relations in design) and they provide an abstraction mechanism to facilitate e.g. the creation, manufacturing or analysis of workpieces or more general to bridge the gap between several systems in the world of Computer Integrated Manufacturing (CIM). Features that are required e.g. for design may differ considerably from those required e.g. for manufacturing or assembly, even though they may be based on the same geometric and technological entities [6].

So representing features is one necessary step to bridge the gap between several CA*-systems and an important step towards truly Computer Integrated Manufacturing. The expected advantages of a close coupling of CA*-systems are:

The information interchange shall lead to a better knowledge transfer, to shorter turnaround times and to improved feedback. At the end, higher flexibility and generally better results are expected.

In current research one method to represent features is based on graph grammars (cf. [3, 6, 15]). This area is a well established field of research and provides a powerful set of methods like parsing, and knowledge about problems, their complexity and how they could be solved efficiently [5]. So in consideration of the feature characteristics 'made-to-measure' tools must be developed to make the recognition and representation process more efficient.

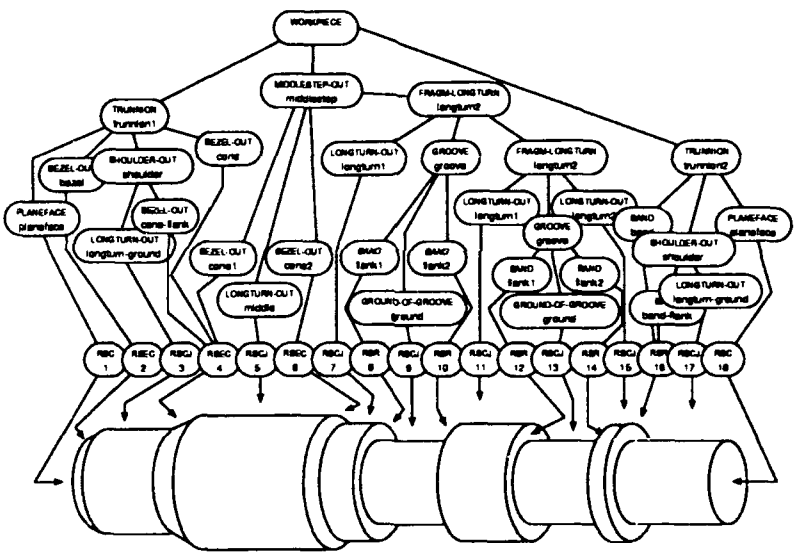


Fig. 1. A workpiece and its feature structure

From this point of view we present in this paper an implementation of the high level domain-specific feature representation language FEAT-REP [10]. This implementation is realized by the Graph Grammar Developer (GGD) – an intelligent tool to support users of FEAT-REP to fill the knowledge base with definitions of features.

2 What are Features ?

To become more familiar with the effect of feature characteristics to our representation formalism, we would like to introduce briefly the most important characteristics of its descriptions. Detailed explanations and the analogue to graph grammars can be found in [10]. Some of the most important syntactical characteristics of features are:

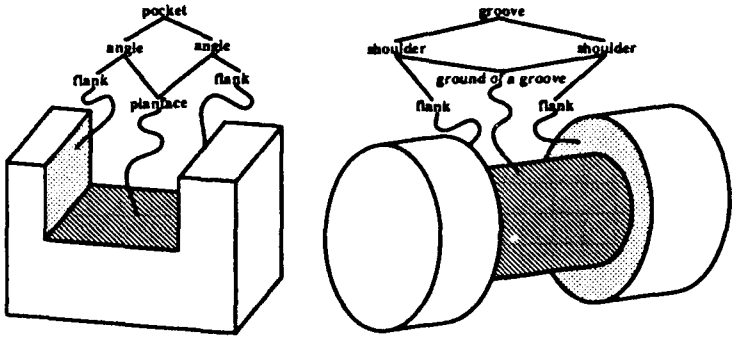


Fig. 2. Overlapping of shoulders or angles

Similar definitions: In an application the knowledge base containing the feature descriptions will be large. Many of these descriptions will be similar to each other, descriptions for a single feature as well as those for various features, with respect to a *has-parts* (e.g. figure 1) and a *is-a* hierarchy.

Component overlapping: Features may have relations to features of different workpieces (e.g. bearing).

Dependence of Dimensions: In dependence of dimensions, the same structures may be identified as different features (e.g. *groove* and *insertion*).

Fragmentizing: Parts of features are not always in direct neighborhood (e.g. *FRAGM-LONGTURN* in figure 1).

Ambiguity: In the terminology of features an expert often have different alternative descriptions for the same structure (e.g. *groove* or *pocket*).

Neighborhood: Feature descriptions form graphs of features and/or surfaces (see figure 3), where edges represents the neighborhood.

Interaction: Areas of features can overlap (see figure 2).

Additional characteristics are *contextsensitivity* (e.g. *LONGTURN-OUT*, *GROUND-OF-GROOVE* in figure 1), and *defectivity*.

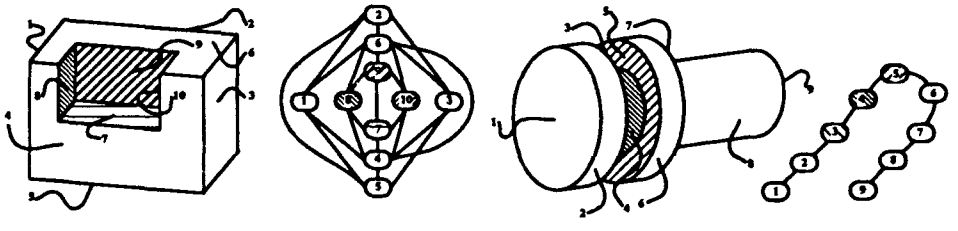


Fig. 3. Neighborhoodgraphs of surfaces

3 Attributed Node Labeled Feature Graph Grammars

In this section we will briefly define the terminology of attributed node labeled graph grammars as used in this paper. Introduction and survey can be found in more detail e.g. in [5].

In our paper the term *(feature-) graph* means an attributed finite undirected node labeled graph, in the sequel shortly called *graph*. Such a (feature-) graph FG is defined as a 4-tupel $FG := (V, E, \Sigma, \varphi)$, where V is a finite (nonempty) set of *attributed nodes*, $E \subseteq V \times V$ is a set of undirected *edges*, Σ is a finite (nonempty) alphabet of *node labels* or *sorts* and φ is a *labeling function*, with $\varphi : V \rightarrow \Sigma$. Workpieces are represented by such graphs. The nodes of a workpiecegraph represent geometric primitive surfaces, the node label decode the type of the surface (e.g. *cylinder jacket*), the attributes carry detailed geometric and technologic information (e.g. tolerances) and the edges decode the topology of the workpiece, i.e. two nodes are adjacent if the corresponding surfaces touch each other.

An *attributed node label (feature-) graph grammar* (ANLFGG) is a 4-tuple $GG := (T, N, P, goal)$, where T is a finite (nonempty) set of *terminals*, N is a finite (nonempty) set of *non-terminals*, P is a finite set of *productions* and $goal \in N$ is the *start node*. A *production* (rule) $p \in P$ is a 4-tuple $(lhs, rhs, \varepsilon, c)$ where $lhs \in N$ is a single node, the *left hand side* of p, rhs is a (nonempty) (feature-) graph over $T \cup N$, the *right hand side* of p, ε is an *embedding specification* and c is a finite set of *conditions* over lhs and rhs, the so-called *dependency relations*. The conditions or the so-called *constraints* c serve two purposes: First to proof or generate informations by calculating attributes and second to lay down certain restrictions and attributes given by a description of a feature.

The most graph grammar formalisms are distinguished by the embedding specification ε. In our case we define ε in that way that always an edge in a (feature-) graph of a derivation step represents the neighborhood of the two incident nodes. For details of our ANLFGG and the analogue to features see [11] and [10].

4 System Architecture of GGD

In contrast to other more general tools editing graph grammars (cf. [7, 9]) the GGD is specialized to edit FEAT-REP – the 'made-to-measure' (feature-) graph grammar formalism. Figure 4 shows the most important components of GGD and their interrelations.

The **visualization** component is the graphical user interface of the GGD. It offers the user an easy possibility to enter, view and manipulate definitions of features (see figure 7). A part of this component is a text-editor (*Constraints for ...*) to enter conditions. Using the designated menus all functions of the other components could be called. The GGD could also be used without taking advantage of the visualization component.

In figure 4 and 5 the visualization of a typical feature is shown. The user may add or delete nodes, neighborhoods and overlaps. For any of the nodes the

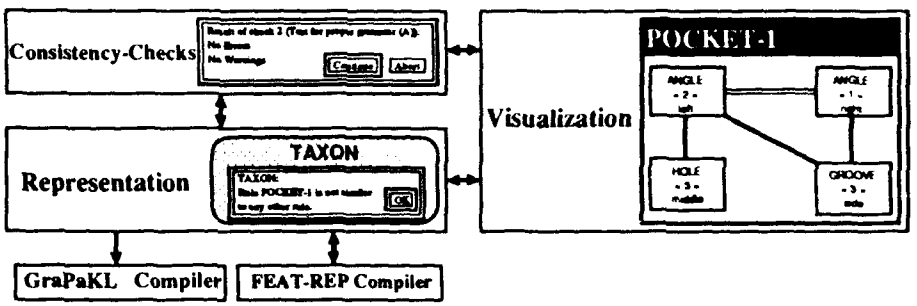


Fig. 4. Structure of the GGD

sort has to be given, a label representing a second more specific name given by the user is optional but useful; the numbers are used for the parser GraPaKL as a kind of heuristics to specify an order in which he will try to find instances for the nodes. So they may change during the lifetime of the specified production. The optional labels support the descriptions of the conditions in a more natural way to identify the nodes in mind. Additional functions are provided to close or resize windows, or to move nodes and edges.

As shown the features are entered as graphs, which is a very abstract way to represent features. To give a more vivid illustration we currently develop a tool to show features as they would look as part of a workpiece. A first prototype is shown in figure 7.

Figure 5 shows the feature *Pocket* and the corresponding features as they are represented by the GGD. Not shown in this illustration are any conditions belonging to the rules.

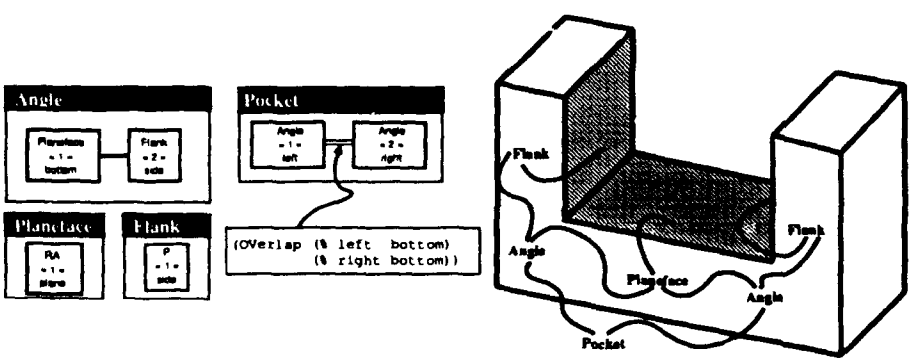


Fig. 5. The rules forming a pocket

The **representation** component stores the entire knowledge. Several functions are provided for access and modification of the (feature-) graph grammar. Integrated in this component is a concept language based on KL-ONE (Knowledge Language ONE), called TAXON [8], which is developed for technical domains.

One drawback which concept languages based on KL-ONE have is that all the terminological knowledge has to be defined on an abstract logical level. In many applications like ours, one would like to be able to refer to concrete domains and predicates on these domains when defining concepts. Examples for such concrete domains are the integers, the real numbers or also non-arithmetic domains, and predicates could be equality, inequality or more complex predicates. TAXON realize a scheme for integrating such concrete domains into concept languages rather than describing a particular extension by some specific concrete domain. The used algorithms such as subsumption, instantiation and consistency are not only sound but also complete. They generate subtasks which have to be solved by a special purpose reasoner of the concrete domain [1].

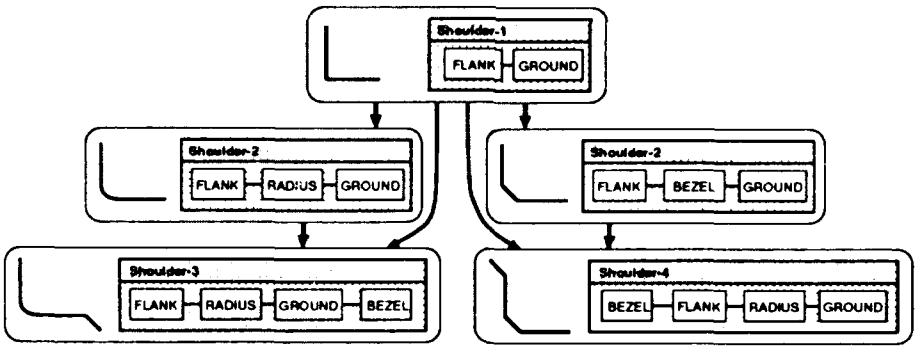


Fig. 6. A hierarchy of features

TAXON is used to handle the feature characteristic of many similar definitions by defining a hierarchy of the productions. The right hand side of any production is compiled to a convenient form to be stored in TAXON. It was necessary to find a representation which allows the concept language to compute exactly the subsumption hierarchy we respectively the expert have in mind. This has to be done efficient as the (feature-) graph grammar may be large. In TAXON a production a subsumes a production b, if b is an *expansion* of a, i.e. b can be generated out of a by inserting nodes into the right hand side of a.

According to the feature characteristic of many similar definitions TAXON hold to kinds of hierarchy: One for all feature definitions and one for every feature. Note that the latter is not just a part of the former.

Figure 6 shows a simple hierarchy of three features. *Shoulder-2* and *Shoulder-*

4 are expansions of *Shoulder-1*, *Shoulder-3* of *Shoulder-1* and *Shoulder-2*. It should be noted that our hierarchy is more extensive than just a subgraph relation. In future work the similarity of conditions will also be taken into account.

The GGD offers several **consistency checks** and verify the defined grammar for soundness. This will be performed during the development of a (maybe new) (feature-) graph grammar. The tests are adapted to our purpose, the aim is to prevent the description of features. This offers the user the possibility to detect and to eliminate the most errors as early as possible. Some of the performed tests are:

- A grammar can't be used without a start node. So it has to be checked if it has been defined and if it appear in any production on the right hand side. In the case of manufacturing or design features this should be a production for *workpiece*.
- Our definition requires that every feature graph is connected. Therefore the system checks if the productions right hand side is connected.
- GGD verifies if the defined grammar sounds [4]. This verification contains the following checks:
  - There is no non-terminal (production) with an empty right hand side.
  - Every non-terminal can be expanded to a terminal graph. (Is there any senseless non-terminal ?)
  - For every non-terminal or terminal the start node can be expanded to a graph containing this symbol. (Is there any unreachable symbol ?)
- The GGD performs the task to check for a correct syntax of the conditions.
- A test is performed if every sort used by a production and its conditions is defined in the associated hierarchy. In addition GGD proof the hierarchy for cycle-free definitions.

Some checks are performed when a new or changed production and its associated conditions are saved by the user to the knowledge base. The complete check (see figure 1: *check rulebase*) is only performed on a request from the user.

The **FEAT-REP compiler** has the capability to read and to write files of this specific graph grammar formalism [10]. These files represent the knowledge base containing the descriptions of features. They are usable by programs for recognizing features (parse) and also by programs for feature based design (generate).

The program Graph Parser KaisersLautern (GraPaKL, [11]) is a heuristic driven chart based parser for our (feature-) graph grammars ANLFGG, adopted to recognize features of workpieces. The **GraPaKL compiler** translates the data stored in the representation component of GGD to files processable by the GraPaKL, say to its internal representation formalism. GraPaKL realize an abstraction step by transforming the geometrical and technological description of a workpiece into the qualitative level of the feature terminology. As result a feature structure is expected (see e.g. figure 1).

5 Developing a Feature Graph Grammar with GGD

The most important components of our graph grammar ANLFGG are the set of productions specifying the feature definitions and a hierarchy of sorts where every production is associated to one sort. If a production in the GGD is defined without specifying the associated sort, GGD automatically prompt an editor for defining it.

To develop a feature graph grammar the following sequence of steps is recommend to be performed:

Define the set of sorts specifying the super- and subsort relations. Querying the consistency check for the knowledge base maybe defined cycles will be found. Additionally GGD will point out, that there are no associated productions.

Define the set of productions. A copy-function can be used to specify similar rules. Also all conditions associated to a production have to be specified. After defining a production, GGD automatically check the (syntactical) correctness of this production. Also it is possible to check the classification of this production by TAXON.

Perform the consistency check for the grammar. After defining the set of sorts and the set of productions, during 4 stages the integrity of the knowledge base is checked. Errors or Warnings are maybe given by GGD.

Save the defined grammar in a FEAT-REP file. This file can be read again by GGD to modify the defined grammar or to generate a file for the parser. So a knowledge base have not to be defined in one session; interruptions are possible even though some errors occur during the previous step. GraPaKL files should be saved only if there are no errors in the knowledge base.

A successful feature graph grammar provided the drawing up (as a kind of a knowledge acquisition step) of a catalog containing the feature descriptions (syntax and semantics) in an informal manner. From one's own experience a typical sketch of the described features make this step more easy and more effective. It is important that this step is performed together with a knowledge engineer or at least by using domain specific acquisition tools (e.g. [13, 16]). After describing the feature graph grammar GraPaKL is recommend to be used for checking the knowledge base of features on concrete workpieces.

6 Conclusion

We introduced an intelligent system to support the representation and the developing of features in CAD/CAM. 'Made-to-measure' graph grammars are used as a formal foundation, which is well suited, to represent the characteristics of features. Our tool GGD to edit our ANLFGG's should be efficient enough to handle even large and sophisticated (feature-) knowledge bases. The computation of hierarchies and the enforcement of several integrity checks make an efficient development of the grammar possible.

The knowledge representation and the integration of TAXON are already implemented. Until today this system is used by our CAPP-system called PIM (Planning In Manufacturing, [12]) to generate and maintain the knowledge base for manufacturing features. But it is also usable as domain independent editor for the specified graph grammar.

Future extension will be additional semantics checks, an improved user interface and a tool to generate graphs representing workpieces. GGD will also be integrated with the editor V-SKEP-EDIT [18] to offer the possibility of describing features and the associated skeletal plans in one session. Also a visualization of the defined features as shapes will be generated in future research. Figure 7 illustrate the today implemented user interface of GGD. In one window the user can highlight the features on the workpiece recognized by GraPaKL - the feature recognizer.

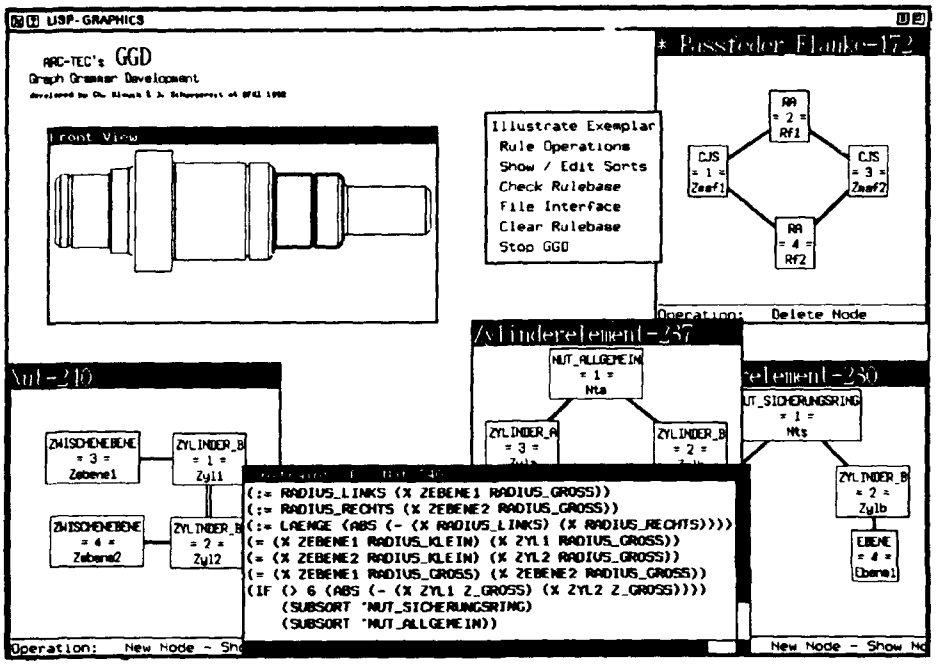


Fig. 7. User interface of GGD

Currently GGD was used by a mechanical engineer to specify design features [17]. The training period takes about one week. No special knowledge about TAXON and the semantics checks was needed. Just the syntax of the language to specify the conditions of the features (*Constraints for ...* Window) which is like COMMON-Lisp takes a little bit time to learn.

7 ACKNOWLEDGEMENTS

This research was performed during the ARC-TEC project (Acquisition, Compilation and Representation of TEChnical knowledge) at the German Research Center for Artificial Intelligence Inc. (DFKI). It has been supported by grant from The Federal Ministry for Research and Technology (FKZ ITW-8902 C4).

References

1. Baader, F. et al: *A Scheme for Integrating Concrete Domains into Concept Languages.* in: 12th IJCAI, pp. 452-457, 1991.
2. Chang, T.-C.: *Expert Process Planning for Manufacturing.*, Addison-Wesley, 1990.
3. Chuang, S.-H. et al: *Compound Feature Recognition by Web Grammar Parsing.*, in: Research in Engineering Design, Springer-Verlag, pp. 147-158, 1991.
4. Drobot, V.: *Formal Languages and Automata Theory.*, Computer Science Press, Freeman & Co., 1989.
5. Ehrig, H. et al: *Graph Grammars and Their Application to Computer Science.*, 1th - 4th International Workshop, Springer Verlag, LNCS 73, 153, 291, 532, 1979-1990.
6. Finger, S. et al: *Parsing Features in Solid Geometric Models.*, in: ECAI'90, pp. 566-572, 1990.
7. Goettler, H.: *Graphgrammatiken in der Softwaretechnik.*, Informatik Fachberichte 178, Springer-Verlag, 1989.
8. Hanschke, P. et al: *TAXON: A Concept Language with Concrete Domains.* in: PDK'91, Springer-Verlag, LNAI 567, pp. 411-413, 1991.
9. Himsolt, M.: *An Interactive Tool for Developing Graph Grammars.* in: 4th International Workshop of [5], pp. 61-65, 1990.
10. Klauck, Ch. et al: *FEAT-REP: Representing Features in CAD/CAM.*, in: IV ISAI: Applications in Informatics, pp. 158-168, 1991.
11. Klauck, Ch. et al: *A Heuristic Driven Parser Based on Graph Grammars for Feature Recognition in CIM*, in: SSPR'92, pp. 200-210, 1992.
12. Klauck, Ch. et al: *Heuristic Classification for Automated CAPP.*, in: 11th AAI-XI'93, pp. *forthcoming*, SPIE, 1993.
13. Kuehn, O. et al: *Integrated Knowledge Acquisition from text, previously solved cases and expert memories.*, in: Applied Artificial Intelligence, vol. 5, pp. 311-337, 1991.
14. Mullins, S. et al: *Grammatical Approaches to Engineering Design, Part I: An Introduction and Commentary.*, in: Research in Engineering Design, Springer-Verlag, pp. 121-135, 1991.
15. Rinderle, R.: *Grammatical Approaches to Engineering Design, Part II: Melding Configuration and Parametric Design Using Attribute Grammars.*, in: Research in Engineering Design, Springer-Verlag, pp. 137-146, 1991.
16. Schmidt, G.: *Knowledge Acquisition form Text in a Complex Domain.*, in: 5th IEA/AIE-92, pp. 529-538, 1992.
17. Schulte, M. et al: *Recognition of Design Features from Product Models.*, in: 9th ICED'93, pp. *forthcoming*, 1993.
18. Wu, Z. et al: *Skeletal Plans Reuse: A Restricted Conceptual Graph Classification Approach.*, in: 7th AWCG, pp. 142-152, 1992.

A Knowledge-based Approach to Group Analysis in Automated Manufacturing Systems

Kesheng Wang

Department of Production Engineering
The Norwegian Institute of Technology
N-7034 Trondheim, Norway

ABSTRACT Knowledge-based techniques has been applied to many fields, including the design and management of manufacturing systems. This paper emphasizes on how to build a knowledge-based system (KBGAS) by the use of XPLAIN programming tool, for group analysis in automated manufacturing systems. Clustering algorithm follows Kusiak's method. This approach is suitable for integrating models and algorithms which have been developed in production engineering with a knowledge based system. The particular system (KBGAS) provides a user-friendly interface and can be operated without any special computer knowledge or knowledge of artificial intelligence.

1. INTRODUCTION

Knowledge-based systems have received considerable attention during the last few years. This interest stems from the recognition that technical knowledge in a flexibly automated, computer-integrated manufacturing systems is increasingly developing into a prominent factor of production. Computer-integrated manufacturing (CIM) embraces both the technological as well as the administrative information flow. Data-producing and data-processing machines are included in a continuous flow of information in order to represent information-needing processes in a transparent, available and nonredundant manner. Computer-aided design, planning, control, manufacturing and quality assurance, as well as knowledge-based systems are to be linked into this information flow (14).

Group Technology (GT) is a philosophy for cellular manufacturing which has been given especially consideration in an automated manufacturing system. Group analysis, a subtask of the group technological approach, is here understood as the task of partitioning the machines of a factory into independent groups, and identifying exceptional machines and products which prevent the task from otherwise being completed successfully (4). Group analysis aims at organizing the production system so that product traffic between different groups, or cells, is minimal. While having several advances in comparison to the use of a functional layout, group technology may also cause problem such as workload imbalance (13).

The group analysis in group technology in automated manufacturing systems can be loosely formulated as follows: determine machine cells and part families with minimum number of parts that visit more than one machine cell, and select a suitable material handing carrier with the minimum corresponding cost subject to the following constraints:

1. Processing time available at each machine is not exceeded.
2. Upper limit on the frequency of trips of material handling carriers for each machine cell is not exceeded.

3. Number of machines in each machine cell does not exceed its upper limit or, alternatively, the dimension (e.g., the length) of each machine cell is not exceeded.

Several algorithms and programs for performing group analysis have been developed. The simple method suggested by Falster (6) finds group in a somewhat unrealistic case where the groups already exist without any modifications to the production system. The graph-theory approach performs the group partition by using clique-finding algorithm.(11) Some methods, e.g., the inter class traffic minimization method attempt to find the partition so that some parts are still manufactured outside their associated groups,(7) while other methods leave some parts and machines completely outside the partition.(8) Further, the partition into groups allow some parts outside their "own" cells, as well as by introducing new machines.(9)

The algorithm and formulation of the GT problem is not only computationally complex, but also involves constraints that are difficult to handle by any algorithm alone. Therefor, we promote the use of a knowledge-based system in the implementation of production system design tools.

XPLAIN is a programming tool available on UNIX workstation using X-window system. It has been developed during the last four years at SINTEF-NTH. XPLAIN has been used to develop several commercial applications to offshore industry.(5)(15) We have also used XPLAIN to develop welding expert system,(2) knowledge-based computer-aided process planning system.(10)(12) During the implementation of XPLAIN, we find that XPLAIN is quit suitable for the purpose of integrating of expert system with industrial environments. In this paper we show how a Knowledge-Based Group Analysis System (KBGAS) is developed by use of XPLAIN, a programming tool, in a flexible automated manufacturing system.

2. SYSTEM ARCHITECTURE

The Knowledge-Based Group Analysis System (KBGAS) architecture, including database, knowledge base, model and algorithm base, inference engine, explanation module, knowledge acquisition module, user interface, is shown in Fig. 1.

2.1 Input Data

The input data required by KBGAS fall into two categories:

- Machine data
- Part data

In addition to these data, depending on the characteristics of the manufacturing system, the following optional data can be provided:

- Maximum number of machines in a machine cell
- Maximum frequency of trips that can be handled by a material handling carrier (for example, AGV and robots)

2.2 Grouping Process

Prior to the beginning of the grouping process, KBGAS constructs a machine-part incidence matrix based on the data provided by the user.

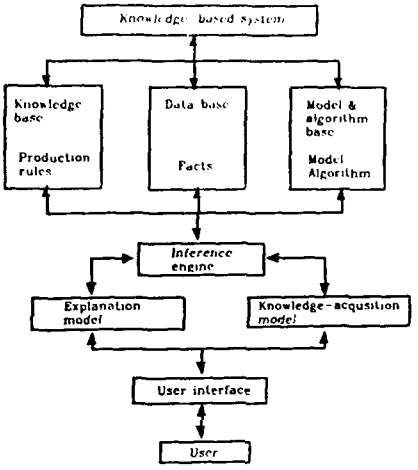


Figure 1. System architecture

Next, the KBGAS initialized in the data base; objects representing facts known about the manufacturing system are considered. Then the system forms machine cells and the corresponding part families. Each machine cell is formed by including one machine at a time. A machine is first analyzed for the possibility to inclusion in machine cell. For example, a bottleneck machine, i.e., a machine that processes parts visiting more than one machine cell, is not included.

Each time a machine cell has been formed, the KBGAS checks whether the constraints 1-3 have been violated and removes all parts violating the constraints.

For a machine cell that has been formed and analyzed by KBGAS, the corresponding machines and parts forming a part family are removed from the machine-part incidence matrix. The system does not backtrack in the grouping process, i.e., once machine cell is formed, the machines included in the cell are not considered for further machine cells.

2.3 Output Data

At the end of the grouping process, KBGAS prints the following data:

- Machine cell formed.
    - machine cell number
    - list of machines in a machine cell
    - part family number
    - list of part number in a part family
- Part waiting list.
- List of machines not used.
- List of bottleneck machines.
- Maximum number of machines in a cell.

3. DATA BASE

The data base stores the basic facts or declarative knowledge currently known about problem domain in the form of objects and frames. Such data are obtained from the user in an interactive mode. The user is required to enter such data as follows:

- Machine frame: Machine frame contains information regarding end machine
- Part frame: The part frame contains information regarding each part
- Matrix-t (machine-part incidence matrix): The machine-part incidence matrix is constructed by the system based on the input data
- Current machine.
- List of candidate machines.
- List of temporary candidate machines.
- Part waiting list.
- List of bottleneck machines.
- List of temporary bottleneck machines.
- List of machines not used.
- MC-k (machine cell k).
- PF-k (part family k).

4. KNOWLEDGE BASE

The knowledge base stores the domain-specified procedural knowledge need to solve problems coded in the form of production rules. The knowledge base consists of six kinds of production rules:

1. Preprocessing rules which deal with the initialization of objects in the data base that are not provided by the user.
2. Current part rules which deal with the current parts being included in part family, for example, whether a current part should be placed in the part waiting list.
3. Current machine rules which deal with the procedure from selecting candidate machine to selecting current machine. selected machine can be put in the candidate machine list and a candidate machine can selected as current machine.
4. Machine part rules which check the appropriateness of a current machine to the machine cell being formed, for example, weather the current machine is a bottleneck machine.

5. Machine cell rules which deal with each machine cell that has been formed. Machine cell rules check for violation of constraints and remove parts violating them.
6. Material handling selection rules which deal with the selection of material handling carriers for a formed machine cell.

Each rules has the following format:

Rule number (**IF** conditions **THEN** actions)

5. THE INFERENCE ENGINE

The inference engine examines facts and implements the rules stored in the knowledge base in accordance with logical inference and control procedure. In this system, a forward-chaining control strategy is employed. In a given class of rules it attempts to fire all the rules that are related to the context considered. If a rule are triggered, i.e., the conditions are true, then the actions of the triggered rule are carried out. However some rules stop the search of the knowledge base and send a message to the algorithm.

6. THE USER INTERFACE

The user interface, representing the interaction between the user and knowledge-based system, provides opportunities for the user to monitor the performance of the system, volunteer information, request explanations, and redirect the problem-solving approach.

7. MODEL AND ALGORITHM BASE

The clustering algorithm developed by Kusiak (6) is an extension of the cluster identification algorithm.(1) This algorithm is stored in the model and algorithm base. It is used to formulate the group analysis problem. The specific model formulated depends upon the nature of the problem, which is reflected in the data provided by the user.

8. IMPLEMENTATION OF KBGAS

8.1 Illustrative example

Give the machine-part incidence matrix shown in Eq. 1, vector fa (frequency of AGV trips required for handling each part), vector fr (frequency of robot trips required to handle each part), max fa = 40 (maximum frequency of trips that can be handled by an AGV), max fr = 100 (maximum frequency of trips that can be handled by robot), and vector T (the column outside of matrix in Eq. 1), solve the group technology problem. The maximum number of machines in a machine cell is 3.

fa [11 30 2.5 - 6 10 - 6 7 15 18 14] max-fa (40)
fr [11 30 5 3 6 15 10 12 7 - 36 28] max-fr (100)

machine number \ Part number	1	2	3	4	5	6	7	8	9	10	11	12	
1	0	4	0	21	0	0	0	8	0	0	0	0	40
2	26	0	5	0	0	10	0	0	0	0	0	0	40
3	0	0	20	0	10	0	0	0	0	22	0	8	40
4	0	35	0	0	0	0	2	6	0	0	0	0	50
5	5	0	0	0	0	6	0	0	25	0	0	0	50
6	0	16	0	10	0	0	3	0	0	0	18	0	60
7	0	0	0	0	1	0	0	0	0	7	0	7	50

(1)

The result is shown in matrix 2:

machine number \ Part number	1	6	9	5	10	12	2	4	7	8	11	3
2	26	10	0	0	0	0	0	0	0	0	0	5
5	5	6	25	0	0	0	0	0	0	0	0	0
3	0	0	0	10	22	8	0	0	0	0	0	20
7	0	0	0	1	7	7	0	0	0	0	0	0
1	0	0	0	0	0	0	4	21	0	8	0	0
4	0	0	0	0	0	0	35	0	2	6	0	0
6	0	0	0	0	0	0	16	10	3	0	18	0

(2)

Three machine cells, MC_1 = {2,5}, Mc_2 = {3,7}, and MC_3 = {1,4,6}, and the three corresponding part families, PF-1 ={1,6,9}, PF-2 = {5,10,12}, and PF-3 = {2,4,7,8,11} have been generated. Part 3 has been assigned to a functional manufacturing facility. Two AGVs and a handling robot tend the three machine cells.

8.2 Implementation

The Knowledge-Based Group Analysis System (KBGAS) is developed by using of XPLAIN programming tool. XPLAIN is a software development tool available on UNIX workstations using X-window system. This tool has been successfully used to develop a welding expert system in SINTEF-NTH. XPLAIN combines features from advanced User Interface Management System (UIMS), expert system, spreadsheet system, sketching tool and database system (see Fig. 2). The layout description of the user interface is done by sketching interactive form and the functionality is defined in expert system like rules. XPLAIN is not code generating system nor en interactive system. When your XPLAIN application is running, you may step into the rule editor, change rules and add rules. Then you may step back into your running application just by pushing a function button.

The system operates through a pull-down menu, by which the user can move between using cursor keys, and then activate the desired module. Figure 3 shows the main menu, which contains 11 items.

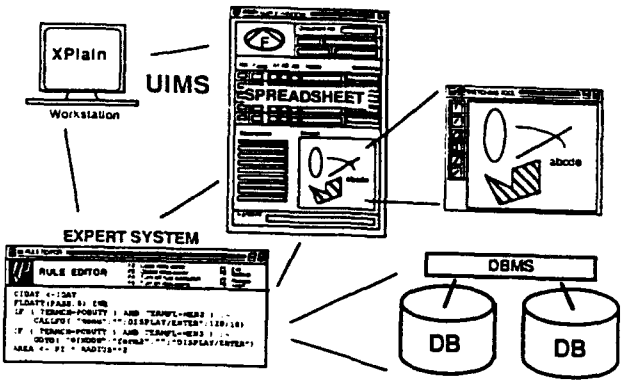


Figure 2. XPLAIN joins feature from different types of programming tools.

The names of the item are self-explanatory. If MANUAL INPUT item is selected, the input data model is activated and interacted with. Figure 4 shows a sub-manu, which further guides the user to input required information, such as, number of machines, number of parts, maximum process time for machines, machine-part matrix, ... etc. Then EXECUTE item can be selected, it handles the input information, interrogates the knowledge base and models and algorithms base, starts inference engine and executes cluster algorithm. Dependent on the items in the main menu, the output can be obtained in three different forms: (1) DATA OUTPUT which is shown in Figure 5. (2) MATRIX FORM which is displayed in Figure 6. and (3) PHYSICAL LAYOUT which is shown in Figure 7.

As shown in Figure 5, for the given example three machine cells and three part families have been developed. MC-1 is served by an AGV, MC-2 is served by a robot, and MC-3 can be served by a robot or an AGV. The overlapping part 3 is placed on the part waiting list. The computation was performed for the maximum cell size equal 3.

9. CONCLUSIONS

In this paper a general formulation of the group technology problem in an automated manufacturing system was presented. The formulation involves a matrix of processing times and three constraints related to the availability of processing time at each machine, requirement for material handling carriers, and the maximum number of machines allowed per machine cell. To solve the grouping problem, a knowledge-based system (KBGAS) was developed by using of XPLAIN programming tool. The KBGAS integrated a heuristic algorithm with a knowledge-based system.

There a lot of models and algorithms in manufacturing engineering

which have been developed and employed. Knowledge-based systems have received considerable attention. To integrate these models and algorithms with a knowledge-based system is an important research area in flexible automated, computer-integrated manufacturing. The system presented demonstrates an approach to integrate some models and algorithms with knowledge-based system by using of XPLAIN programming tool.

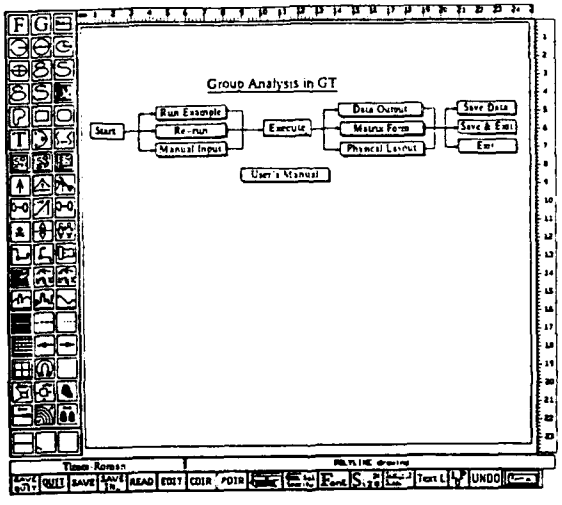


Figure 3. The main menu.

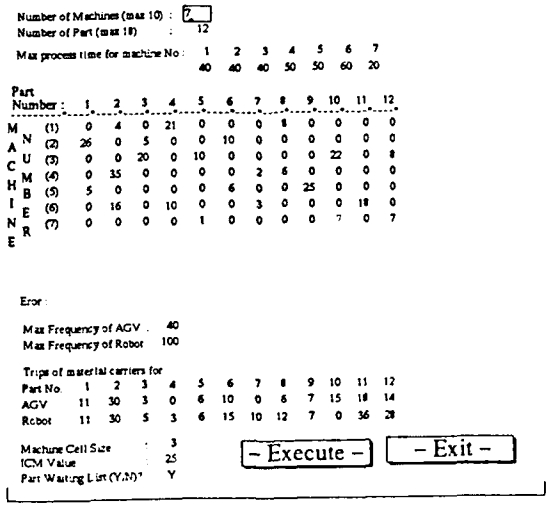


Figure 4. The menu for input matrix.

MC Layout

Machine Cell (1) M3 M7
Part Family (1): P5 P10 P12
MHS Alternative: (AGV)

Machine Cell (2) M6 M1 M4
Part Family (2): P2 P4 P7 P11 P8
MHS Alternative: (Robot)

Machine Cell (3) M5 M2
Part Family (3): P1 P6 P9
MHS Alternative: (Robor or AGV)

Part Waiting List P3
Bottle Neck Machines : No
Number of machines : 7
Number of parts : 12
Machine cell size 3

Figure 5. The menu for data output

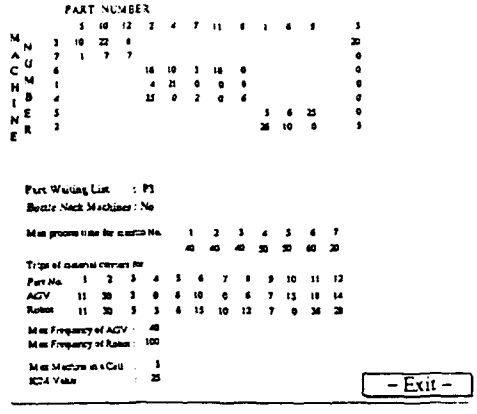


Figure 6. The menu for output in matrix form.

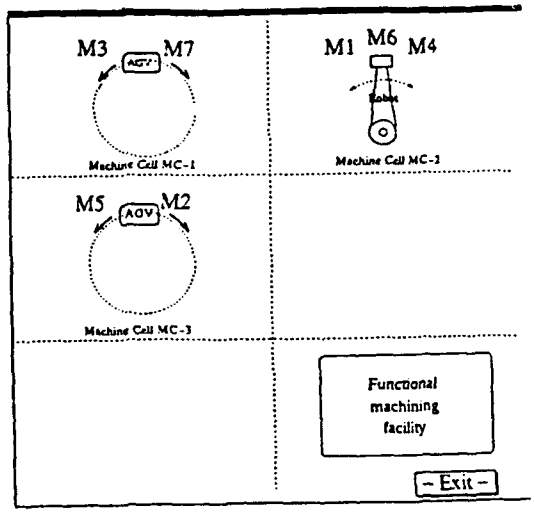


Figure 7. the menu for the physical layout.

10. REFERENCES

1. Anderberg, M. D. (1973). Cluster analysis for application, Academic Press, New York.
2. Bratli, A., Engh, E. and Røstadsand, P. A., The use of computer system for generation of welding procedures and quality control in fabrication of offshore structures and pipework, Proceedings of EUROJIN 1 (First European Conference on Joining Technology), Strasbourg, France, pp. 103-112, 1991.
3. Brooks R. A., Programming in common LISP, John wiley & sons, New York, 1985.
4. Burbidge, J.L., Production Flow analysis, Clarenden Press, Oxford, 1989.
5. Engh, E., SIMWELD - a system for cost simulation, Proceeding of EUROJIN 1, Strasbourg, France, pp. 35-41, 1991.
6. Falster, P., Structural techniques for the design of production systems, Advances in production management system, North-Holland, pp 23-42, 1986.
7. Harhalakis G., Ioannis M. and Rakesh N. Development and application of knowledge based system for cellar manufacturing, Proceedings of the third international conference expert systems and leading edge in production and operations management, Columbia, USA, pp. 343-355, 1989.
8. Kusiak A., A knowledge based system for group technology. International journal of production research, vol. 26, no. 5, pp. 887-904.
9. Lassila, O., Knowledge-based algorithm for group analysis, Advances in production management systems, Elsevier science publishers B. V. (North-Holland), IFIP, pp. 507-513, 1991.
10. Mykiebust, o., Knowledge-based process planning with object oriented implementation, Complex machining and AI-methods, NORTH_HOLLAND, pp. 49-58, 1991.
11. Rajagopalan R. and Batra J. L., Design of cellular production systems - A graph-theoretic approach. International journal of production research, vol. 13, no. 6, pp. 567-579, 1975.
12. Romstad, A., User requirement specification for the process planning supervisor, version 2, SINTEF, 1990.
13. Shambu, G, Ramaswamy, R. and Rao H. R., A rule-based system for scheduling in a hybrid group technology environment, Proceedings of the third international conference expert systems and leading edge in production and operations management, Columbia, USA, pp. 357-367, 1989.
14. Spur, G. and Specht, D, "Knowledge-based diagnosis in manufacturing systems", Manufacturing Systems, Vol. 19, No. 2, 1990.
15. Wold, P., Experience from development and implementation of QCWELD - A computer-aided NDT planning, documentation and inspection system, Proceedings of EUROJIN 1, Strasbourg, France, pp. 215-226, 1991.

CENTER: A System Architecture for Matching Design and Manufacturing

Bei-Tseng Bill Chu and He Du
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC 28223 (billchu@unccvax.uncc.edu)

ABSTRACT

This paper presents CENTER, a new architecture for design for manufacturing. CENTER is based on a new methodology for matching product design with manufacturing processes. The power of CENTER derives from a new technique to analyze manufacturing data and use knowledge-based techniques to guide design simulations based on the result of the analysis. CENTER can be used to construct knowledge-based systems that would compliment the prevailing practice of exclusively relying on human-directed design simulations. We present two case studies: application of CENTER to a welding process and application of CENTER to VLSI manufacturing.

1. Design for manufacturing

Matching product design with the manufacturing process is the key to the success of statistical process control. To understand the this concept, let's first consider a simplified view illustrated in Figure 1. Suppose the quality of a product is determined by parameters x and y. The box (also referred to as the process window) in figure 1a shows the ability to control these parameter in actual manufacturing; that is, that the vast majority (e.g. 99.9999998%, or 6-σ) products can be made with x and y within the box. The design engineer must then make sure that the design would work with any parameters in the process window. Determining the area in the parametric space where a particular design would work, indicated by a dotted boundary in figure 1, is typically difficult. Simulation is the principle tool used to verify designs. It is impossible to simulate every point within the process window. One typically simulates the "worst case scenarios" and hoping that the area in the parametric space where the design would work is larger than the process window (as illustrated in Figure 1a).

If the area in the parametric space where a design would work is larger than the process window, we then say that design and manufacturing are *centered*, as illustrated by Figure 1a. In such a case the manufacturing process would produce very few defective parts (3 defective parts per million if a process is under 6-σ control). On the other hand, figure 1b shows a case where the design and manufacturing are not centered. In this case, even the vast majority of parts made are within the process window (box), many of them will still fail because of the design.

In this paper we define *design sensitivity* as the different between the process window and the area where the design actually works. Figure 1a has no design sensitivity, the left hand corners of the box in figure 1b are design sensitivities. Even with the great care exercised by design engineers, design sensitivities do occur. Its prevalence in integrated circuit industry is well documented [9]. It also occurs in other

[1]This research is supported by a grant from the National Science Foundation MIP-9017151

processes such as welding. Discovering design sensitivities is a very difficult problem, especially in cases where a non-negligible amount of random defects exist. The CENTER architecture presented in this paper is designed to construct intelligent assistants in helping identifying design sensitivities.

2. The CENTER system architecture

The CENTER architecture is based on the idea that feed back from manufacturing data can be used to intelligently guide simulations to center design and manufacturing. Suppose a part is made with parameters x^0 and y^0, within the process window, and the part is judged to be defective. One of the following two must be true. First, this is due to *design sensitivity*, or the design will always fail when parameters take values x_0 and y^0 (such as one of the left corners of the box in figure 1b). Second, the failure is due to some uncontrolled factor(s), or *random defect*. A statistical model has been developed [2] to distinguish the two possible cases. The goal of CENTER is first to use such a statistical model to hypothesize design sensitivities (i.e. occurrences with high probability of being design sensitivities) and then use knowledge-based techniques to target specific simulation runs to verify such hypotheses. Design engineers would benefit from CENTER by learning unexpected design sensitivities.

Figure 2 illustrates the CENTER architecture. This architecture is designed to address the common need to center design specification in manufacturing process windows. The goal is to isolate those modules that can be shared across manufacturing domains and provide an architecture in which such generic modules can work with other domain specific modules. The generic module in the CENTER architecture is the design-sensitivity hypothesis module formulated in the next section of this paper. Hypotheses proposed by the design-sensitivity hypothesis module is verified by a domain specific knowledge-based system. If the hypothesis is rejected, the hypothesis module may reformulate its hypothesis. Thus CENTER shares the principle of hypothesize-and-testing with many intelligent systems [8].

3. Hypothesis of design sensitivities

Throughout the rest of paper the term parametric space refers to a (multi-dimensional) region where manufactured parts will take values from. In other words a parametric space represents the capability of the manufacturing process. The approach we take can be stated as:

If one has observed that there is a region in the parametric space containing failed parts only, and the probability of this event occurring due to random defects is very small, then one can hypothesize a design sensitivity in this region of the parametric space.

This strategy can be achieved in two steps. First, we look for a region in parametric space containing failed parts only. Second, we construct a probability model for random defects and perform a statistical hypothesis testing.

3.1. Hypothesizing a region in the parametric space

There are many ways one can define the boundary of a group of failed dies in a d-dimensional parametric space. Our aim is concentrated on looking for a region suspected of having a design sensitivity; expert investigation will be relied upon to

determine the root cause of the design sensitivity. Therefore, using Occam's Razor, we are only concerned with convex regions, where each surface of the boundary represent some linear combination of parameters that may cause design sensitivity. The simpliest algorithm would be the Convex Hull method [4]. However, because we are dealing with many points in a high dimensional space, the complexity of the algorithm is too great to be practical.

We use an alternative greedy algorithm using a series of linear discriminant surfaces [3]. Too start with, we have a d-dimensional hypercube representing the process window. Figure 3(a) shows a cube representing the specification for three parameters, failed parts are depicted as dark dots. The simpliest case involves finding a linear surface separating the passed parts from the failed dies as indicated in figure 3(b). This surface along with the the boundary surfaces of the parametric space specification forms the convex region of interest.

More generally, the method of Two-Category Linear Discriminant is to find a Decision Surface that cuts a given space in two parts. The Decision Surface can be represented as $g(\mathbf{x})=0$, where $\mathbf{x}$ is a vector of variables of the parametric space, $g(\mathbf{x})$ is a linear discriminant function:

$$g(\mathbf{x})=\mathbf{w}^t\mathbf{x}+w_0 \quad (1)$$

where $\mathbf{w}$ is vector of constants called the *weight vector* and w_0 is the threshold weight. The two partial spaces separated by $g(\mathbf{x})=0$ are $g(\mathbf{x})>0$ and $g(\mathbf{x})<0$.

Typically, we cannot completely divide all failed and passed dies using one discriminate surface. However, a series of such surfaces will define a convex region enclosing a group of failed die.

More specifically, let

$$\mathbf{Y}=\begin{bmatrix}1\\ x_1\\ \cdot\\ \cdot\\ \cdot\\ x_d\end{bmatrix}=\begin{bmatrix}1\\ \mathbf{x}\end{bmatrix} \qquad \mathbf{A}=\begin{bmatrix}w_0\\ w_1\\ \cdot\\ \cdot\\ \cdot\\ w_d\end{bmatrix}=\begin{bmatrix}w_0\\ \mathbf{w}\end{bmatrix}$$

then

$$g(\mathbf{x})=\mathbf{A}^t\mathbf{Y} \quad (2)$$

Consider n parts with position vectors $\mathbf{Y}_1, \mathbf{Y}_2, \cdots, \mathbf{Y}_n$ in the parametric space. We would like to construct a surface $g(\mathbf{x})$ minimizing the perceptron criterion function [3]. A succession of such linear surfaces will define a convex hull with failed parts only.

3.2. Differentiating random defects and design sensitivity

With enough manufacturing data, one expects that parts failed due to design sensitivities will be localized to certain areas of the parametric space while parts failed due to random defects will permeate the entire parametric space. However one would like to detect design sensitivities early in the development process, the challenge is to find a probability model that would lead to a statistical test to differential random defects from design sensitivities using as small amount of data as possible.

We use the following formulation to describe a generic manufacturing concern. Suppose that parts are manufactured in *lots*. Each lot contains W slots. Each slot consists of K positions. Each position contains a part we are manufacturing. This

characterization can be used to model both batched as well as discrete manufacturing processes. In a discrete manufacturing case where one part is manufactured at a time, W=K=1.

Suppose the parametric space is d-dimensional, and that L lots have been manufactured. Values for all d parameters of all T=L×W×K parts have been measured. Each part has been tested for pass/failure. Suppose that F of the total T parts failed.

Each manufactured part can be regarded as a point in the d-dimensional space. Suppose a convex region X with N defective parts is found in the parametric space. Let the probability of X occurring due to a design sensitivity be denoted as p_{ds}. Let $p_r=1-p_{ds}$ be the probability of X occurring due to random defects.

We now demonstrate how p_r can be estimated. Suppose we take the position that there are no design sensitivities, and the failure of parts are independent. Then the probability for having N failed parts in region X is

$$p_X=(\frac{F}{T})^N \quad (3).$$

Since there are a total of T parts, the problem becomes a binomial distribution: we have $\frac{T}{N}$ regions; the probability for each region to contain all failed parts is p_X; the probability of finding at least on region with all bad parts due to random defect is:

$$p_r=1-(1-p_X)^{T/N} \quad (4).$$

However the real-world cases are more complicated in that the failure of parts may not be independent. In the most general case we expect three types of dependencies. *Lot dependency* refers to cases where parts made within the same lot tend to fail together. *Slot dependency* refers to cases where parts made within the same slot tend to fail together. *Cluster dependency* refers to cases that parts made within the vicinity of each other in the same slot tend to fail together. Cluster dependency differs from slot dependency in that parts made within the same slot but sufficiently apart from each other may still fail independently. The idea is to find the *largest* dependent group (lot, slot, or cluster) based on observed data. Assume that failure within a group is dependent and failure across groups are independent, or

$$p_X<(\frac{F}{T})^{\lfloor N/Q \rfloor} \quad (5).$$

Combining (4) and (5) we have:

$$p_r<1-(1-(\frac{F}{T})^{\lfloor \frac{N}{Q} \rfloor})^{\lceil \frac{T}{N} \rceil} \quad (6)$$

To determine the largest dependent group, one starts by looking for evidence of *lot-dependent defects* (this step is not necessary if the lot size is one). We assume that in the absence of *lot-dependent defects*, the success rate for each lot follows a normal distribution (binomial distribution if the lot size is small). Well known statistical tests [1] exist to verify whether observed data follow such a distribution. If there is evidence for *lot-dependent defects*, we would like assume that Q equals to the lot size.

If, on the other hand, we fail to find evidence for *lot dependent defects*, we then look for evidence of *slot-dependent defects* (again skip this step if slot size is one). We again base such a test on the assumption that in the absence of *slot-dependent defects* and *lot dependent defects*, slot success rates are expected to follow a normal distribution (binomial distribution if slot size is small). If evidence of *slot-dependent*

defects is found, we take Q to be the number of parts per slot.

If evidence supports neither *lot-dependent* nor *slot-dependent defects*, we have to consider the cluster's size. If the number of parts per slot is one, then Q is one. Otherwise cluster size varies with manufacturing processes. Process dependent knowledge is needed to estimate Q. It is common to use Γ distribution to model cluster size [5].

4. CENTER-VLSI: VLSI circuit manufacturing

This section describe a prototype system, CENTER-VLSI, we have built for identifying design sensitivities in VLSI circuit manufacturing based on the CENTER architecture. CENTER-VLSI uses the generic design sensitivity identification module of figure 2. A number of VLSI circuits (parts) are fabricated together on a wafer (slot). Wafers are organized into lots. The selection of cluster size is based on [5].

Figure 4 depicts a typical methodology to design a VLSI circuit. A VLSI circuit typically involves over one million transistors. It is impossible to simulate the entire circuit based on electrical parameters (e.g. mobility, threshold voltage). Instead a design engineer would select a set of electrical parameters from the parametric space defined by the manufacturing process. These parameter values are used to simulate basic logic elements (e.g. gates, flip flops) and identify their delay characteristics (figure 5 shows some typical delay characteristics for a flip flop). Then the entire circuit is simulated at logical level to verify the design.

The dimensionality, proportional to the sophistication of the technology used, is typically very high (greater than 10). Resource constraints dictates that one only selects a very small fraction of the parametric space, referred to as the worst case scenarios, to conduct simulations according to figure 4. Selecting such scenarios is an art practiced by only the most experienced engineers. Even then, design sensitivities are difficult to avoid, especially for early design versions, because it is difficult to foresee all potential interactions among the large number of transistors.

The CENTER architecture is ideally suited to attack such a problem. We describe how a prototype system, CENTER-VLSI, addresses this problem domain under the CENTER architecture. The basic idea of CENTER-VLSI is to have a program initiate all simulations to verify the design sensitivity discovered by the design sensitivity identification module.

CENTER-VLSI, its organization shown in figure 6, relies on a simulation case base acquired from the design engineer. The case base is indexed by the name/version of the design. When an engineer performs a SPICE [7] simulation (based on electrical parameters), with the help of the engineer, this circuit input (in a format readable by SPICE) is recorded in the case base. Delay characteristics extracted from the SPICE simulation is also recorded. Typically the design engineer uses an automated tool to construct logic circuit design based on simulated circuit fragments and their delay characteristics. The role of CENTER-VLSI is that of a learning apprentice [6] in this phase of design.

When manufacturing data is collected and a design sensitivity region, X, has been hypothesized, the simulation driver of CENTER-VLSI will randomly select several points from X and repeat the simulations recorded in the case base. Complete logic simulation will also be performed based on the newly obtained delay characteristics under the frequency where actual circuit failed. If the design fails under a set of electrical parameters hypothesized by X, then X would be identified as a design

sensitivity and the result is presented to the engineer for design revisions. Otherwise, the simulation driver will set all parts in X as passed and reinvoke the design sensitivity hypothesis module.

Due to the difficulties in obtaining real design/process data, we tested CENTER-VLSI with a detailed simulated experiment. The experiment has two parts. In the first part we design a circuit with a known design sensitivity. In the second part we test to see if CENTER-VLSI can discover this "bug" on its own.

In the first part, we follow the design methodology of 4 but avoided selecting any electrical parameters from the known design sensitive region. The circuit we use is a Self-Voltage Controlled Oscillator circuit (figure 7). The reasons for selecting this circuit is that circuit's frequence is very sensitive to device parameters. It is clear that for such a simple circuit, it is unlikely for a trained design engineer to commit the type of design mistake we introduce on purpose. The idea being tested here is that the same type of design sensitivities (e.g. in terms of the combination of circuit parameters) could have been committed in a much more complicated circuit due to unforeseen interactions.

The circuit has two components: an Oscillating Route and a Voltage Reference. The Oscillating Route has 31 stage CMOS inverters and three voltage controlled NMOS transmission transistors. The Voltage Reference is a differential amplifier with a built-in voltage divider as its input. Both parts are sensitive to parameters, especially to the threshold voltages and gain factors of two type devices.

In this example, we selected the following electrical parameters: TOX,VTON,VTOP,KPN,KPP. TOX is gate oxide thickness for both NMOS and PMOS devices; VTON and VTOP are NMOS and PMOS threshold voltage (vsb=0) respectively; KPN and KPP are NMOS and PMOS gain factors respectively. The assumption is that we are building this circuit with only one size of transistor. In a more realistic situation, different sizes of transistors will be used at the same time and the number of electrical parameters will increase accordingly.

We select the MOSIS process (a public fabrication process funded by the National Science Foundation) as our reference manufacturing process. We have obtained a set of approximate process specifications for these five parameters, as well as other parameters for our simulations.

We first divide the specified ranges of these five parameters into five intervals. Assuming all other parameters being constant, and selecting mid values from each divided intervals, we have total of 5^5 (3125) possible parametric combinations. We use SPICE3e2 to simulate the frequency performance at all these combinations leading to a a frequency distribution in a five dimension parametric space. We define 35MHz as the lowest acceptable frequency for this design, thus the region in the parametric space corresponding to frequencies under 35MHz is a design sensitivity.

We proceed to define a simulated manufacturing process that would assign electrical parameter values to each "manufactured" part (a five-place vector). However, precise distributions for device parameters in real manufacturing is difficult to get. To carry out our experiment, we made the following simplifying assumptions. To a certain extent, these assumption can be justified based on the statistical characteristics of device parameter distributions.

(1) The parameter average value for a given lot follows a Gaussian distribution.

(2) The parameter variance within a lot is smaller than the variance among lots. Furthermore, parameter variance of a lot follows a Gaussian distribution.

(3) The parameter average value within a wafer in a given lot follows a Gaussian distribution for VTON,VTOP,KPN,KPP. The mean value of TOX for a wafer is assumed to be uniformly distributed within the limits for a lot.

(4) The parametric variance withing a wafer is smaller than that within the lot containing that wafer. This variance also follows a Gaussian distribution.

(5) Within a wafer, the distribution of parameters is linear for TOX, and concentric for VTON,VTOP,KPN, and KPP.

Defects on wafers are also generated randomly. We use a uniform distribution to simulate *point defects*. We use a Gamma Distribution model [5] to simulate *clustered defects*. To simulate *wafer-dependent defects*, we use a uniform distribution to determine whether a wafer is subject to *wafer-dependent defects* or not. If it is subject to such defects, we again uniformly determine dies on that wafer that are subject to such *wafer-dependent defects*. To simulate *lot dependent defects*, we assume each lot has equal probability of being subject to *lot-dependent defects*. Once a lot is selected as being subject to a *lot-dependent defect*, we random determine what dies would fail for a wafer, and assumes all wafers within the lot will have the same failure pattern due to the *lot-dependent defects*.

A total of 30,000 parts where "fabricated" using our simulated process. The total failure rate is set at 60%, a typical situation on a pilot line. About 300 parts fall into the design sensitive region. CENTER-VLSI is able to correctly identify a subset of the design sensitive region containing 100 failed parts.

5. CENTER-WELDING: identifying design sensitivity in a welding process

To demonstrate that the CENTER architecture can be applied to different application domains, we illustrate how it can be used in a generic welding process [2]. The main parameters of this process are: the speed of the work piece, the feeding speed of the welding material, the voltage and current applied by the welding gun. The workpieces are selectively X-rayed for defect inspection. Defects can be caused by either process parameters or by other random effects such as impurity of the welding material. The CENTER architecture can be applied to identify welding conditions that will lead to defects in the workpiece.

6. Summary

This report presents CENTER as a system architecture for matching design and manufacturing. At this writing, main components of CENTER-VLSI has been implemented. We are actively seeking opportunities to apply CENTER-VLSI to real design/fabrication data.

7. Reference

(1) Box, G, Hunter, W, & Hunter, J, *Statistics for Experimenters*. New York, NY: Wilely, 1978.

[2] Due the proprietary nature of this process, we cannot present details of our experiment.

(2) Chu, B. & Du, H. *Identifying Design Sensitivities Based on Fabrication Data*, Dept. of Comp. Sci. Univ. of North Carolina at Charlotte, Technical Report 92-4-1, April, 1992.

(3) Duda, R. & Hart, P. *Pattern Classification and Scene Analysis*, John Wiley & sons, 1973.

(4) Edelsbrunner, H. *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

(5) Michalka, T., Varshney, R. & Meindl, J. "A Discussion of Yield Modeling with Defect Clustering, Circuit Repair, and Circuit Redundancy" in *IEEE Tran. on Semiconductor Manufacturing*, Vol. 3. No. 3. pp.116-127, August, 1990.

(6) Mitchell, T., Carbonell, J., & Michalski, R. (eds.) *Machine Learning* Boston, MA: Kluwer Academics Publishers 1986.

(7) Nagle, L. "SPICE2: A computer program to sumulate semiconductor circuits" Memo No. ERL-M520, University of California at Berkeley, 1975.

(8) Reggia, J. *Knowledge-Based Decision Support Systems: Development Through KMS*, Dept. of Comp. Sci. Univ. of Maryland, Technical Report, TR-1121, Oct., 1981.

(9) Spence, R. & Soin, R. *Tolerance design of electronic circuits* Reading, MA:Addison-Wesley, 1988.

(10) Sze, S. *VLSI Technology*, New York, NY:McGraw Hill, 1982.

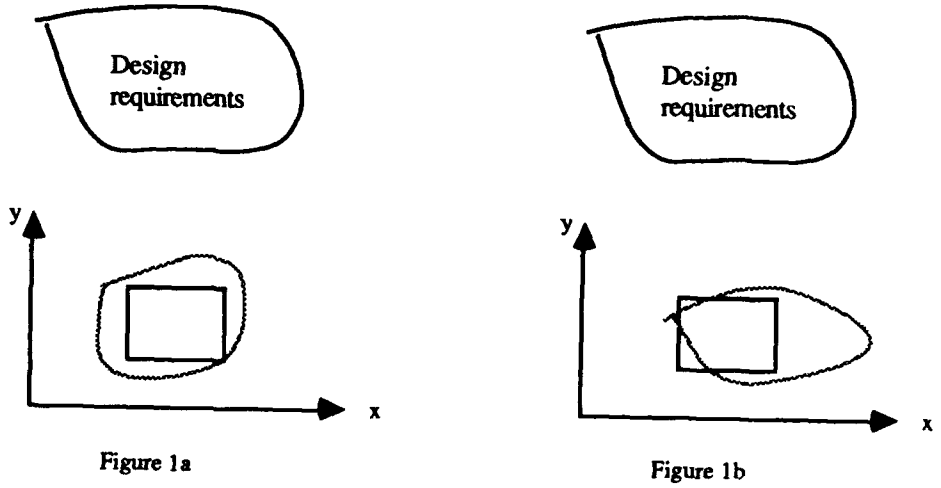


Figure 1a

Figure 1b

Figure 1. Matching design and manufacturing

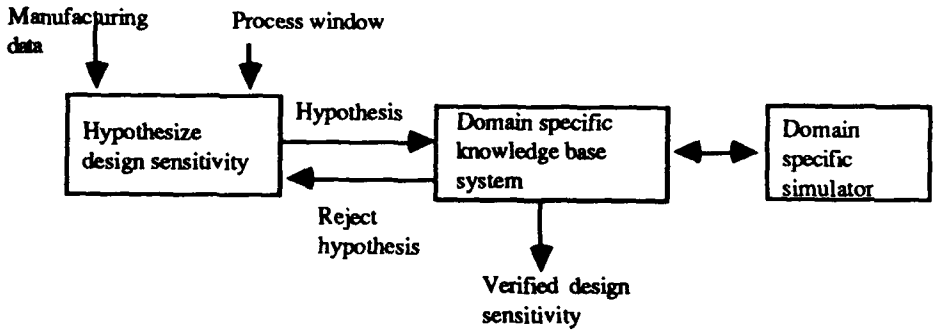


Figure 2. CENTER architecture.

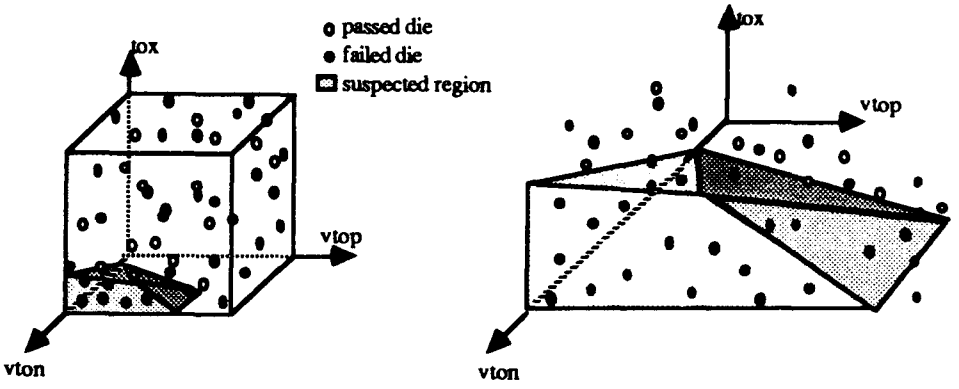


Figure 3(a) 3-dimensional parametric space Figure 3(b)

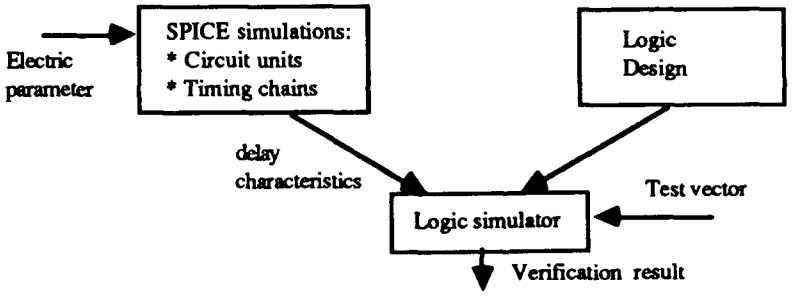


Figure 4. A typical VLSI design methodology

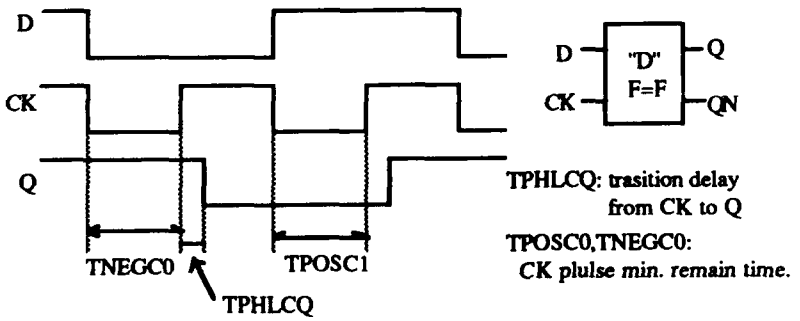


Figure 5 "D" Flip-Flop delay characteristics

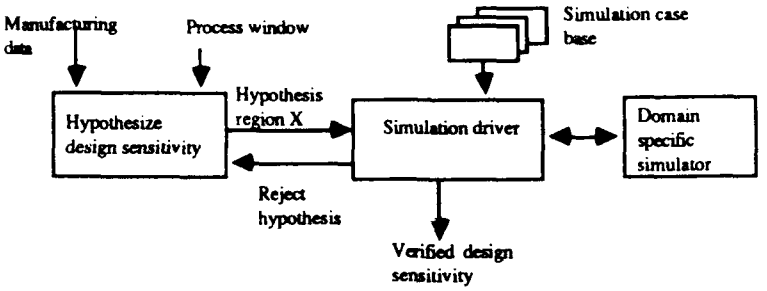


Figure 6. CENTER-VLSI system diagram

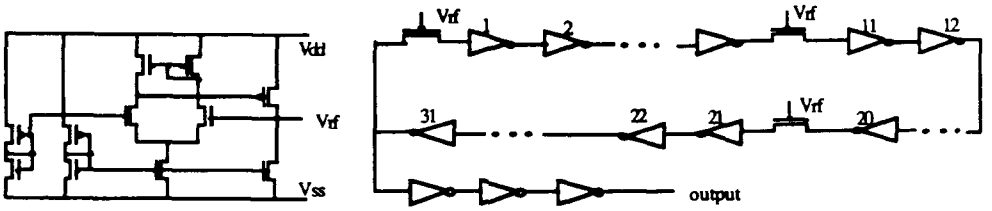


Figure 7 31-stage CMOS Oscillator

Knowledge-Based System Integration in a Concurrent Engineering Environment*

M. Sobolewski

Concurrent Engineering Research Center, West Virginia University, Morgantown 26506
E-Mail: sobol@cerc.wvu.wvnet.edu

Abstract. The systematic integration of humans with the tools, resources, and information assets of an organization is fundamental to concurrent engineering (CE). In an integrated environment, all entities must first be connected, and they then must work cooperatively. Services that support "concurrency" — through communication, team coordination, information sharing, and integration — in an interactive and formerly serial product development process provide the foundation for a CE environment. Product developers working concurrently in their application domains need "built-in" tools and "operated-in" tools in a computer-aided CE environment. The latter group of tools evolves over time and requires continuous extensions and changes since each next product development is either new or an improvement of the previous one. Product developers need a CE programming environment in which they can build programs from other developed programs, built-in tools, and knowledge bases describing how to perform a complex design process. This paper describes a type of system integration provided by means of a knowledge-based environment that encompasses programs, CAx tools, and knowledge bases. The presented approach is illustrated by selected examples.

1 Introduction

System integration, many consider, is an ill-structured problem (the term *ill-structured problem* is used here to denote a problem that does not have an explicit, clearly defined algorithmic solution). No specific rules have to be followed when doing integration; integration depends totally upon the environment to be integrated. Experienced designers deal with system integration using judgement and experience. Knowledge-based programming technology offers a methodology to tackle these ill-structured integration and design problems. The Concurrent Engineering Center (CERC) has developed such an environment, called *DICEtalk* (Sobolewski, 1990a,b, 1991a,b,c, Kulpa and Sobolewski, 1992).

DICEtalk is a knowledge-based development system developed for the DARPA Initiative in Concurrent Engineering (DICE), implemented in *Smalltalk* object-oriented environment as an implementation tool for engineering design, modeling systems, and system integration (Goldberg and Robson 1989). It includes a knowledge definition and problem solving apparatus together with a set of state-of-the-art user interface tools. It is capable of handling multiple interacting hierarchical knowledge

*This work has been sponsored by the Defense Advanced Research Projects Agency (DARPA), under Grant No. MDA972-91-J-1022 for the DARPA Initiative in Concurrent Engineering (DICE).

bases and integrating external (foreign) programs with the knowledge base mechanism. It also supports natural language knowledge definition, fully menu-driven user interaction, and graphical data presentation. More than 20 knowledge-based engineering tasks were experimentally implemented using the system and integrated into a design process model (Padhy & Dwivedi, 1990; Padhy, 1990; Sobolewski, 1990b, 1991a,c; Saidi, 1991; Benner 1991; Chung 1992).

If we assume that everyone in the product development cycle will make the best decision from the overall life-cycle viewpoint, if they are given appropriate advice at the right time, then we can attempt to design a set of knowledge-based systems to provide such advice. The knowledge-based integration framework requires the capture of expert knowledge and analytical tools to build the knowledge-based environment, which will support each of product developers involved in new product design, development, prototype, and manufacture. Properly integrated knowledge-based systems provide many substantive benefits, including an intuitive interaction paradigm, transparent access and invocation of tools integrated, the capability to share relevant data among tools and services, and the capability to combine tools' capabilities to provide for compound transactions.

We now briefly review some terminology. If a particular set of facts is known about the world, then this *factual* (*declarative*) *knowledge* can be increased if various *rules* (*norms*) are known. Factual knowledge may be derived using both *observed facts* and *derived facts* through a justified mode of inference. In this simple characterization, the term *declarative knowledge base* is taken to mean the collection of all facts of the world (*domain*), and the term *inference engine* refers to programs that *reason* with (*execute*) that declarative knowledge base. An *inference engine* derives facts (output facts — *conclusions*, and intermediate facts — *findings*) from other facts, which include rules, assumptions, user answers, findings, etc. A collection of programs and subroutines needed to compute new facts and conclusions, or used during the execution of a declarative knowledge base by a procedural attachment mechanism, forms a *procedural knowledge base*. This procedural attachment mechanism can provide graphics, windows, animation, dictionary lookup, or file I/O. Both the declarative knowledge base and the procedural knowledge base are referred to as a *global knowledge base*, or simply a *knowledge base*. A knowledge base requires an inference method and fact and goal representations. In knowledge-based systems, these two parts constitute a knowledge representation paradigm.

In the rest of this paper we describe the DICEtalk representation scheme, which involves several language levels. We also delve into the problem solving aspects supporting system integration. An object-oriented problem solving scheme is based on a "dispatching-managing" model. Finally, we discuss specific examples.

2 DICEtalk Knowledge Representation Scheme

In DICEtalk, a knowledge description scheme is based on a surface language, an intermediate language, and a deep language, with the internal languages hidden from the user (Sobolewski, 1987, 1989a, 1989b, 1991b). The surface language sentences appear as simplified English sentences that allow engineers to create knowledge bases and metaknowledge bases naturally, without requiring them to learn specialized data description and manipulation languages. The expressions of intermediate language are logical formulas of the formalized percept language for describing percepts and metapercepts (Sobolewski, 1989b, 1991b). In this case, language primitives of surface

sentences and percept formulas are the same, i.e., a subject of a sentence and its complements, which allow the conversion of natural sentences into percept formulas to be easier and more natural. The deep language is Smalltalk-80 used for implementing structured objects that represent logical (percept and metapercept) formulas at a computer level (Goldberg and Robson, 1989).

The knowledge description language *SPDL* (Surface Percept Description Language) is used to express declarative (factual) knowledge and metaknowledge bases, as well as goal knowledge bases, whereas the programming language *Smalltalk-80* is dedicated to representing procedural knowledge bases. SPDL is defined by 44 EBNF (Extended Backus-Naur Form) rules. In order to give a general idea of what SPDL is, we list below 12 basic SPDL rules:

1 *sentence* = *assumption* | *rule* | *goal* | *initData* | *question*.
2 *assumption* = [*entry*] *clause*.
3 *goal* = [*entry*] *clause*.
4 *rule* = [*entry*] *"IF" clause "THEN" clause*.
7 *clause* = [[*"not"*] *subject*] *complements*.
8 *subject* = *path term* ":" | *inputVariableConstant path* ":".
9 *path* = *attributeName* {*attributeName* | *variableName*}.
10 *complements* = *complement* {(",", | *"and"* | *"or"*) *complement*}.
11 *complement* = *path* (*"is"* | *"are"* | "=") *term* [*definiteness*] |
path [*"not"*] *decisionAttributeName* [*definiteness*] |
outputVariableConstant [*"is"* | *"are"* | "="] *path* |
(*"if"* | *"whether"*) *decisionAttributeName inversePath* |
"no" attribute | *"has" attribute* | *predicate*.
14 *term* = literal {(*"and"* | *"or"*) *literal*} | *number* | *interval* | *point* | *vector* |
date | *time* | *multivalue* | "[" *SmalltalkExpression* "]".
15 *literal* = [*"not"*] (*valueName* | *variableName*).
17 *predicate* = "[" *booleanExpression* "]" |
"[" *SmalltalkExpression "for" variableName* "]".

These rules explain the declarative and procedural knowledge-base integration that is based on expressions included in brackets in Rules 14 and 17. Generally speaking, a clause is a description of an entity in terms of subjects and complements, as in natural sentences. Subjects represent main qualities and quantities, whereas complements represent complementary qualities and quantities. Qualities are expressed by paths — sequences of attributes, and quantities by values, i.e., numbers, intervals, points, names, etc. The following example describes the boring machine HBM2:

boring machine HBM2:
table area is 1000,
surface finish is 63,
tolerance is 0.01

where each quality is expressed by a sequence of two attributes or one, the main quantity is a name, and complementary quantities are numbers. Logical connectives (*and, or,* and *not*) are allowed to build compound complements and values, including variables. When a subject is omitted in a clause, it means that it is understood.

SmalltalkExpression stands for any sequence of statements (expression series) of the Smalltalk language, possibly with variables, including the special variable "kb",

which represents the current knowledge base of the DICEtalk system (a knowledge base is an instance of class **KnowledgeBase** or its subclass); *booleanExpression* denotes *SmalltalkExpression* which evaluates to true or false (boolean objects). These two forms of Smalltalk expressions provide a procedural attachment mechanism. This mechanism is especially essential to engineering design tasks, in which many pieces are given in the form of formulas and calculations, and are most effectively carried out by appropriate procedures or external programs.

3 Problem-Solving

A problem-solving model is a scheme for organizing reasoning steps and domain knowledge to construct a solution to a problem. The central issue of problem-solving deals with the question: What pieces of knowledge should be applied, and where and how should they be applied? A problem solving model provides both a conceptual framework for organizing knowledge and a strategy for applying that knowledge. The DICEtalk problem-solving model is object-oriented and based on a dispatch-managing problem-solving model. We can view such a dispatch-managing model as a natural study of how a group of individual solvers can combine to solve a goal (problem). The presented approach is to split the goal into simpler tasks and to solve each of these tasks by a *dispatch-managing module* (DM module). A dispatch-managing module consists of a task *dispatcher* and its *manager*. We suppose that tasks are not independent, i.e., they interrelate in some way.

The dispatch-managing model deals with problem-solving by separating a goal into a hierarchical structure of subordinate tasks solved by DM modules. A dispatch-managing architecture of the problem-solving engine consists thus of five basic components:

1. The *knowledge* (five panels):
   The main repository of goals, facts, procedures, and control advices.
2. The *supervisor* (master panel):
   The master DM module deals with solving the user defined goal. It creates the top DM module and controls a DM network activity.
3. The *DM network* (dispatcher and manager panels):
   The problem-solving tasks are organized into the hierarchical structure related to the current state of goal-solving. Each local DM module deals with local task-solving, according to its local control strategy and a local knowledge-base taken as a subtask perspective of the knowledge base. Managers of local DM modules are responsible to their dispatchers for local strategy. Managers decide what actions to take next for their dispatchers. Communication and interaction among parent DM modules and their child modules take place through their dispatchers.
4. The *working memory* (data panel):
   Subtasks are created by dispatchers and are scheduled to be solved by their managers according to control advices. DM modules produce changes in the working memory that lead incrementally to a global solution as a unification of all local solutions. Parameters are user-defined characteristics evaluated by dispatchers and then used as arguments of procedure calls.
5. The *results* (inference panel):

Problem solving results (answers, findings, and conclusions) are created by the supervisor and dispatchers and can be transferred to other knowledge bases as the part of their descriptive panels for distributed problem solving.

These five components form a DICEtalk global knowledge base implemented by the highly structured class **GlobalKnowledgeBase**. An instance of this class can be considered a kind of object-oriented distributed blackboard (Jagannathan, 1989). DICEtalk can contain many concurrent global knowledge bases during complex cooperative problem solving, as shown in Figure 1. All DICEtalk agents have the same structure and can request and provide services to and from each other (peer-to-peer processing). A *superagent* is an agent that may create and delete its agents.

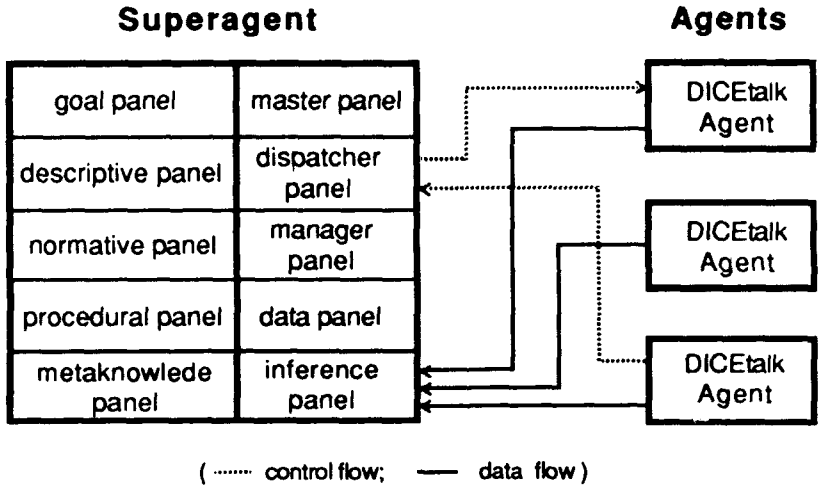


Figure 1. A conceptual view of DICEtalk peer-to-peer processing

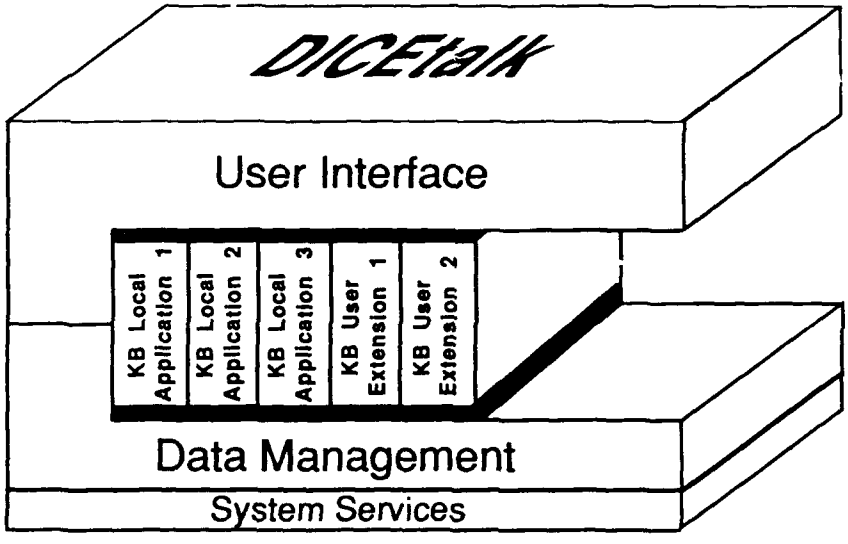
The DICEtalk object-oriented framework of the presented model is implemented by three types of dispatchers: atomic-dispatcher, and-dispatcher, and or-dispatcher; and three types of managers: task-manager, rule-forward-manager, and rule-backward-manager. The problem-solving engine introduces an architecture that treats a declarative and a procedural knowledge base as one active object. Fundamental to this object-oriented integration is the notion of a DICEtalk knowledge base as an instance of a class created when a user defines a new knowledge-based application. This new class, say **SubKnowledgeBase**, is a subclass of the predefined **KnowledgeBase** class. Facts and goals of a declarative knowledge base are stored in instance variables of the **KnowledgeBase** class, whereas instance variables and user defined methods of the **SubKnowledgeBase** class form a procedural knowledge base. Thus, this procedural knowledge base inherits a declarative knowledge base from its superclass **KnowledgeBase**. Procedures defined as methods of the procedural knowledge base can be called by the inference engine when tasks to be solved contain messages sent to *kb*, the current DICEtalk knowledge base.

4 Knowledge-Based System Integration

The following three knowledge-based applications show how declarative and procedural knowledge can be integrated within a single knowledge base and within an integrated system consisting of other developed programs, built-in tools, and knowledge bases describing how to perform a real, complex design process. This three include:

1. The Printed Wiring Board Manufacturability Advisor (PWBMA) (Padhy & Dwivedi, 1990),
2. The Turbine Blade Fabrication Cost Advisor (Saidi, 1990), and
3. The Process Modeling Knowledge-Based System (a Process Modeling Project at CERC, Sobolewski).

The open DICEtalk integration framework is illustrated in Figure 2. This framework is based on three fundamental properties: *homogeneity* of knowledge-based interactions, information exchange, and extensibility.



- Procedural knowledge layer (knowledge wrapper)

Figure 2. The DICEtalk open system framework

4.1 Printed Wiring Board Manufacturability Advisor

The tasks in PWBMA are organized into a hierarchical structure. The main goal of the system is to *evaluate mfg index*. The following rule used in backward reasoning shows the structure of the main goal in terms of its tasks:

IF

border index is x1

and manufacturing index is x2

and process index is x3
and [kb mfgIndex for y]

THEN

mfg index is y

ast1: parameter is Iborder, ast2: parameter is Imanufacturing , ast3: parameter is Iprocess

where *Iborder*, *Imanufacturing*, and *Iprocess* are user-defined parameters (for antecedent subtasks denoted by *ast* with indices) for *border index is x1*, *manufacturing index is x2*, and *process index is x3*, respectively. When this rule is used in backward direction, and the tasks *ast1*, *ast2*, and *ast3* are solved, then their dispatchers assign values of variables *x1*, *x2*, and *x3* to the parameters *Iborder*, *Imanufacturing*, and *Iprocess*, respectively. Now, these values can also be used in the procedural knowledge base as pool variables in methods of the current knowlege-base class. Therefore, when the last antecedent task [*kb mfgIndex for y*] is executed, the message *mfgIndex* is sent to *kb*, the current knowledge base invoking execution of the corresponding Smalltalk method. Within this method, the parameters *Iborder*, *Imanufacturing*, *and Iprocess* defined above in the declarative knowledge base can be accessed as Smalltalk pool variables instantiated by the inference engine:

mfgIndex
^(Iborder value + Imanufacturing value + Iprocess value) / 3

The result of the message is then used as the value substituted for the variable *y*, and the inference engine returns the task *mfg index is y* appropriately instantiated as a conclusion.

Tasks in PWBMA consist of other compound or elementary subtasks. At the bottom of the task hierarchy, elementary tasks are solved by rules, such as the following one for the task *border index:*

IF

board border allowance is x1
and [x1 < 1.0]
and [kb comment: 'the border allowance should be 1 inch']

THEN

border index is D

The attribute *index* takes values *A*, *B*, *C*, and *D* interpreted in the procedural knowledge base as numbers 4, 3, 2, and 1, respectively. In the above rule, the antecedent second task ([*x1 < 1.0*]) and the third task ([*kb comment: 'the border allowance should be 1 inch'*]) are performed as procedural tasks performed by the Smalltalk compiler called by the dispatchers for these tasks.

4.2 Turbine Blade Fabrication Cost Advisor

- Consider the following rule in the Turbine Blade Fabrication Cost Advisor:

IF

cluster blade number is x1
and pattern assembly time is x2
and cluster dress time is x3
and cluster inspection time is x4

and [kb kpp for y]

THEN

pattern wax cost is y

ast1: parameter is Ncb, ast2: parameter is Tpa, ast3: parameter is Tcd, ast4: parameter is Tci

where *Ncb, Tpa, Tcd, Tci* are user defined parameters for *cluster blade number is x1, pattern assembly time is x2, cluster dress time is x3,* and *cluster inspection time is x4,* respectively. Again, when this rule is used in backward direction and the tasks *ast1, ast2, ast3,* and *ast4* are proved, then the inference engine assigns the values of variables *x1, x2, x3,* and *x4* to parameters *Ncb, Tpa, Tcd,* and *Tci,* respectively. Next, when the task *[kb kpp for y]* is executed, then *kpp* is sent to the current knowledge base. The result of that message is used as the value substituted for the variable *y,* and then the inference engine returns the task *cost wax pattern preparation is y,* appropriately instantiated as a finding. The message *kpp* is implemented as the following Smalltalk method:

```
kpp
    | kppCost |
    kppCost := 1 / Ncb value * (Rpl value + Tpa value + Tcd value + Tci value)
        * kc1Cost + kpCost.
    tbfCost := tbfCost + kppCost.
    ^kppCost asFloat
```

where *tbfCost, kc1Cost and kpCost* are instance variables (for other costs) in the procedural knowledge base, and *Ncb, Rpl, Tpa, Tcd,* and *Tci* are parameters defined above in the declarative knowledge base for the rule and treated as Smalltalk pool variables in the procedural knowledge base and therefore in the method *kpp.*

As the above two examples illustrate, the presented scheme of declarative and procedural knowledge representation is based on the following mechanisms:

1. Transfer of data from the inference engine to the procedural knowledge-base via user defined parameters associated with antecedent tasks in rules.
2. Procedure calls by the inference engine to execute tasks in the form *[booleanExpression].* If the variable *kb* appears in *booleanExpression,* it means that methods of the current procedural knowledge base are executed (SPDL Rule 17).
3. Transfer of data from the procedural knowledge base to the inference engine by executing tasks in the form *[SmalltalkExpression for variableName]* (SPDL Rule 17).

4.3 Process Modeling Knowledge-Based System

The DICEtalk distributed knowledge-based environment also is supported by the above mechanisms. Consider a knowledge base *machining* in the Process Modeling Knowledge-Based System that contains the following SPDL tasks, as well as a knowledge base about these tasks:

1. Evaluating design modifications

2. Evaluating machining costs for design features
3. Evaluating machining processes

Each of the tasks itself has an appropriate knowledge base. If the first task is selected to be solved, then the following rule is to be fired:

IF
[*kb solveWith*: '\usr\kbs\design']
THEN
design modifications is evaluated

and the task [*kb solveWith*:'\usr\kbs\design'] is executed. The method *solveWith:* is the standard DICEtalk method defined in the **KnowledgeBase** class as follows:

solveWith: aString
`Switch to a global knowledge base with name aString and open Problem Solving Browser on it. Close Problem Solving Browser opened on the receiver's global knowledge base.`

The new DICEtalk agent for the knowledge base *design* (to be formed in a disk directory \usr\kbs) is created with the following tasks:

1. Specifying surface coordinates
2. Evaluating profile coordinates refinements
3. Specifying superimpose TRUCE code
4. Specifying cutter path TRUCE code

If the first task is selected to be solved, then the following rule is to be fired:

IF
[*kb inputFile*: '\usr\kbs\design';
cd: '\usr\kbs\tools *call:* 'tur';
turOutput: 'file2.dat';
accumulate: #findings *as:* #assumptions]
THEN
surface coordinates are evaluated

and the four messages are sent to the current knowledge base. The methods *cd:call:* and *accumulate:as:* are the standard DICEtalk methods defined in the **KnowledgeBase** class as follows:

accumulate: inSymbol *as:* outSymbol
`Add all elements of the current global knowledge-base collection with the type inSymbol to the elements of the next global knowledge-base collection with the type outSymbol.`

cd: aPath *call:* aString
`Change directory to path name aPath and call a routine aString there.`

The two methods *inputFile*: and *turOutput:* form a kind of a DICEtalk wrapper for the FORTRAN program named "tur" and are defined by the user as part of the procedural knowledge base.

5 Conclusions

This paper has presented a system integration based on a knowledge-based paradigm. A knowledge base is treated as one active object consisting of both declarative and procedural knowledge bases. During problem solving, data is exchanged bidirectionally between declarative and procedural knowledge. The feasibility of using a knowledge-based paradigm in developing systems and system integration for a CE environment has been demonstrated. In the DICE (DARPA Initiative in Concurrent Engineering) program, more than twenty knowledge-based engineering tasks were experimentally implemented with the system: Duct connection (1), Flange connection (2), Material fatigue (3), Disk forging (4), Robot assembly advisor (5), Turbine blade fabrication cost advisor (6), Printed board manufacturability advisor (7), Printed board assemblability advisor (8), Assembly planning simulator (APS) (9), Design process model (PROCESS) (10), Machining processes (11), Preliminary Design Evaluation Module (PDEM) (12), Cost estimation for machining processes (13), Manufacturing process planning (14), Machine operation advisor (15), Machine selection advisor (16), Tool material type selection advisor (17), Tool material selection advisor (18), Cutting fluid selection advisor (19), Knowledge-Based PARametric Finite Element Modeler for aircraft engine blades (KB-PARFEM) (20), and Knowledge-based MODal Finite Element Modeler (MODFEM) (21).

Task 10 integrates all the tasks into a design process model allowing designers to navigate between relevant knowledge bases and engineering tools during problem solving. Task 10 includes Task 11 as a subtask, which itself includes Tasks 12 and 14 as its subtasks. Task 12 integrates four *FORTRAN* programs, and Task 13 integrates two *FORTRAN* programs within the DICEtalk knowledge bases. Tasks 15, 16, 17, 18, and 19 are implemented as knowledge bases integrated as subtasks within the knowledge base of Task 14. Task 20 implements a knowledge base about finite element mesh calculation (integrating here a bunch of external C programs and Smalltalk methods) and then calls the tool *I-DEAS* to do the final finite element analysis. Task 21 implements a knowledge base for geometric parameter selection, substructuring selection, and finite element modeling/analysis (integrating here a collection of external C programs and Smalltalk methods) and then calls I-DEAS to do the final substracturing analysis.

The successful implementation of the experimental applications with DICEtalk suggests the practical usability of the system. Applications were easy and fast to implement, and the user interface and overall system functionality were well received by the users. All the applications mentioned above have been integrated into a design process model under one unifying umbrella to provide an intuitive, uniform model of interactions.

References

Benner, J.C. (1991). KB-PARFEM: A Knowledge-based Parametric Finite Element Modeler for Aircraft Engine Blades. Master's Thesis, West Virginia University, Morgantown.

Chung, S. (1992) Knowledge-Based Mechanical Design Using Substructuring in a Concurrent Engineering Environment. Doctoral Dissertation, West Virginia University, Morgantown.

Du, B., Rachakonda, S., Dwivedi, S., Karinthi, R., Sobolewski, M., and Dax, R. (1992). Forging Process Design in a Concurrent Engineering Environment. J.P. Hager (ed.) *EPD Congress 1992*, A Publication of The Minerals, Metals & Materials Society, pp. 531-545.

Goldberg, A., and Robson, D. (1989). *Smalltalk-80: The Language*, Reading, MA: Addison-Wesley.

Kulpa, Z., and Sobolewski, M. 1992. Knowledge-directed Graphical and Natural Language Interface with a Knowledge-based Concurrent Engineering Environment. *Proc. of the 8th Int. Conference on CAD/CAM, Robotics and Factories of the Future'92*, Aug 1992, Metz, France, pp.238-248.

Padhy, S.K., and Dwivedi, S.N. (1990). A Knowledge Based Approach for Manufacturability of Printed Wiring Boards. *Proc. of the Fifth Int. Conference on CAD/CAM, Robotics and Factories of the Future'90.*

Padhy, S.K. (1990). A Knowledge-Based System for PWB Manufacturability in Concurrent Engineering Environment. Master's Thesis, West Virginia University, Morgantown.

Saidi, M. (1991). Turbine Blade Investment Casting Cost Advisor Model. *Proc. of AACE's 1991 Annual Meeting*, Seattle, WA.

Sobolewski, M. (1987). Percept Knowledge-base Systems. I. Plander (Ed.), *Artificial Intelligence and Information - Control Systems of Robots*. North-Holland.

Sobolewski, M. (1989a). EXPERTALK: An Object-Oriented Knowledge-based System. I. Plander (Ed.), *Artificial Intelligence and Information-Control Systems of Robots*. North-Holland.

Sobolewski, M. (1989b). Percept Knowledge Description and Representation, *ICS PAS Reports* No. 663. Institute of Computer Science of Polish Academy of Sciences.

Sobolewski, M. (1990a). Percept Knowledge and Concurrency, *Proc. of the Second National Symposium on Concurrent Engineering*, West Virginia University, Morgantown., pp. 111-137.

Sobolewski, M. (1990b). DICEtalk: An Object-Oriented Knowledge-Based Engineering Environment. In: *CAD/CAM, Robotics and Factories of the Future '90*, Vol. 1: *Concurrent Engineering*, Berlin: Springer-Verlag, pp. 117-122.

Sobolewski, M. (1991a). Object-Oriented Knowledge Bases in Engineering Applications. *CAD/CAM, Robotics and Factories of the Future '91* Southbank Press, Vol. 1, pp. 470-475.

Sobolewski, M. (1991b). Percept Conceptualizations and Their Knowledge Representation Schemes. Z.W. Ras and M. Zemankova (Eds.) *Methodologies for Intelligent Systems, Lecture Notes in AI 542*, Berlin: Springe-Verlag, pp. 236-245.

Sobolewski, M. (1991c). Integration of Declarative and Procedural Knowledge in Engineering Applications. *Expert Systems World Congress Proceedings*, Pergamonn Press: New York, Vol. 3, pp. 1816-1823.

A Reflective Strategic Problem Solving Model

Patricia Charlton

School of Mathematical Sciences
Bath University
Claverton Down
Bath BA2 7AY
England
pc@maths.bath.ac.uk

Abstract. We define a strategic model which uses reflection as part of its control framework. A strategy is a process of building up units of knowledge sources. We use Newell's [5] control model to give structure. The control structures are classed as unit and part-of-unit processes. The unit is the production statement and the process is how the rule will be applied. The part-of unit is the reflective control for the clustering and building of knowledge sources. This method assists in providing clarification during the process of abstraction for the problem. Our blackboard interpreter, which is used to provide part of the control, is extended in a similar way to OPM [8], using heterarchical abstractions. As the strategic model is developed to provide a reasoning reflective system, we show how the interpreter is extended to provide an adaptive learning system.

1 Introduction

In this paper we give a brief outline of our blackboard interpreter, and show how to extend the structure to give a reflective model for building problem-solving strategies. The application implemented using our model is a forest fire fighting [2] simulation. We show the changes in the blackboard model to explain how to deal with abstraction: that is, problem definition and solution formation. We use fine-grain knowledge source structures to assist in problem representation and abstraction clarity. The reflective process provides control [10] in a complex and changing environment. The problem domain represented for the reflective process, is that of strategies for problem solving. One approach to problem-solving is planning and the planning process requires abstraction which is used to fill in the vague outline of the strategy. In order to reason about the problem, the system is required to provide the properties of strategies, planning and definition in a recursive form.The problem solving model described here provides a structure for defining such problems. First, however, we will describe the blackboard interpreter.

2 The Blackboard Interpreter

Often in AI, there is difficulty in clearly identifying how the problem should be defined in a given system and the problem definition starts to depend on the

system being used. Blackboard interpreters have been found to be difficult to use to build strategic planners for general cases [3]. We support and amplify these findings: when using such systems we have found the control to be restrictive. Strategies require no restriction in the problem definition, but control has to be developed over time.

The blackboard system provides only a general restrictive and uncontrolled environment. This is close to the strategy context but we need to refine the blackboard structure offered so that complex problems can be handled with an appoach which has strategic control. The revised blackboard model gives the necessary control for an adaptive learning system using a strategic method.

The blackboard model requires a problem abstraction for the knowledge sources. The first level of abstraction is to define areas which classify the system. The application used for this discussion is forest fire fighting [2]. A simple abstraction of the knowledge sources for the problem are bulldozer, fire and environment. There are further abstractions which can be considered such as the control centre. It is possible to abstract many knowledge source levels for this application. This helps to illustrate the fine grain knowledge source (KS) definition problem identified by Tate [15], and the abstraction hierarchy difficulties outlined by Craig [3]. Although the system can be defined within the blackboard interpreter, problems are highlighted when trying to represent a dynamic application. Below, we show the template for a simple bulldozer knowledge source requiring a move rule.

Specifying a KS for the Bulldozer Domain

```
Knowledge-Source-Name                    KS-Bulldozer
Trigger-Condition                        $EVENT-L=1
                                           or
                                         $ENTRY=fire
Precondition                               t
Condition-Action pair: Rule-id           Move
                       Condition         $ENTRY=fire
                       Action            Move-Bulldozer
(Local-Variables ())
```

Each rule is given an identifier which is matched to a user-defined event and events can have states which can be changed via the activated knowledge sources as a result of posting an entry by means of the agenda. Once on the blackboard, the entries can be modified by the knowledge sources, providing the agenda allows the change.

The agenda allows the control to determine which knowledge source will contribute to the problem. The knowledge source is executed as a knowledge activation record and may contribute to forming the problem solution. The KSs are static structures while the knowledge source activation records (KSARs) are dynamic and can be modified. It is the entries which are posted onto the blackboard which allows KSs to be *triggered*. Once the triggered KSs, which are represented by KSAR's, become *eligible* they can be executed. The KSARs may add to the blackboard a new event, or a modified event attribute, or add an event attribute. The action part of the knowledge source may be built from a structure

called *propose*, (actions do not have to be a *propose* structure). The *propose* structure causes the events to be modified and created. These event activities may cause changes to the blackboard entries. Below we show an example of a *propose* structure.

```
(PROPOSE :change-type 'add
         :entry 'create-fire
         :level ($VALUE fire-level)
         :attribute 'fire-temperature
         :value (compute-temperature ($GET 'create-fire'fire-position)))
```

The purpose of this action is to calculate the temperature of the area which is now on fire. The fire-level is set by the trigger function and **compute-temperature** is an external application function.

Events are changes to the blackboard state and are caused via an entry being posted. Enteries exist on one level of abstraction on the blackboard, at a given time. They are able to change levels via an event. Network of entries are created as a solution forms on the blackboard. The networks are formed between entries through explicit representation of relationships and the links between entries are through the attribute-value structure.

The communication and the agenda are essentially the control. The control structure says how and when a knowledge source may add to the blackboard. Such an agenda structure is mainly problem-dependent when using the blackboard interpreter [5]. This occurs as the blackboard model is described as an abstract structure [5] and as such becomes problem-dependent as the organistaion framework is implemented. This can cause the model to be less adaptive and domain dependent as discussed by Craig [3]. There are other problems encountered by the control mechanism for updating the blackboard [11]. We propose a networking structure for developing a number of levels. These are mainly developing the agenda, blackboard updating and the reflective process. The strategic model overcomes some of the restrictions in the blackboard model. To achieve an adaptive model we need to add another level of control.

3 Defining the new model

Planning is a method of controlling and monitoring changes in the state of a *problem environment*. There are states which are either static or dynamic. These states are used to represent partial problem abstractions of the system (the system being the problem definition). In the blackboard every KSAR executed is a state change. The user-defined events for the blackboard allow an event to change, the actions being *modify*, *add* or *new*. The strategic networking structure is defined to allow the user to define powerful control systems easily. Control structures work on critical state-changes as a priority mechanism. To achieve control, levels of state inference are defined in terms of unit and part-of unit processes. This becomes important to a reflective or general problem solving system as perception and problem description of the objective can be presented in context.

3.1 The problem context

To give emphasis to a problem context requires an appreciation of the problem in terms of itself i.e. the sub-system needs to be detached while still being part of the problem. This again falls well in to an orthogonal system which is reflective and infinite. Problem organisation hierarchy requires survival towards the problem, the context, and the approach.

The reflective process has no strict hierarchical control but allows sub-components in terms of understanding the interaction that component has i.e. the components provide the part-of unit process for the sub-components. The next level of reasoning [9] informs the system how that control should be handled without breaking the *code* of the strategy i.e. it should be flexible to avoid destruction which is a control to achieve a result.

For reflective process we define the meta-state which gives meta-state level control. The meta-objects, which define the reasoning rules [6], are needed to decide if we can solve the problem at this level. However this can be seen as one process, as Smith [12] discusses the use of identifying the components and realising the structure. The decision process looks after the state process which in turn looks after the decision process.

The agenda is defined as the meta-ks and looks after the other KS structures which have become part-of the agenda structure (Meta-Meta-Ks will look after the Meta-ks etc). The definition will depend on what level the system is being viewed from. All planning structures are defined within themselves [16] so that sub-plans are themselves planning. System adaption is allowed by copying KS to new structure to modify this structure. The next time the agenda uses the structure the level of interpretation has changed. It is a meta-state change which will cause a new KS to be expressed.

3.2 Extensions Required to provide a flexible Strategic Model

Figure 1 shows the explicit panels of the strategic model which is defined in terms of the blackboard structure but is extended explicitly to deal with reflection in terms of the unit process, part-of process and the strategic system.

The *Agenda-ks* is the intermediate process which provides the part-of unit for the *Ks-fine-grain* process unit. The *Agenda-ks* is expressed by the knowledge source template and provides for further structures the same template. The *Meta-agenda* is formed from the *Agenda-ks* and can be a unit which has both *Ks-fine-grain* and *Agenda-ks*. The part-of unit exists as the *Meta-agenda* once it has spawned the *Meta-meta-agenda* process, which follows the same procedure as the *Agenda-ks*. The system as described can view problem definitions and solutions in the form of the blackboard, the agenda and the knowledge source. This allows the different strategic processes to be viewed as different reasoning context definitions. The most reflective process is the part-of which is used to provide the adaptive learning context for the strategic model.

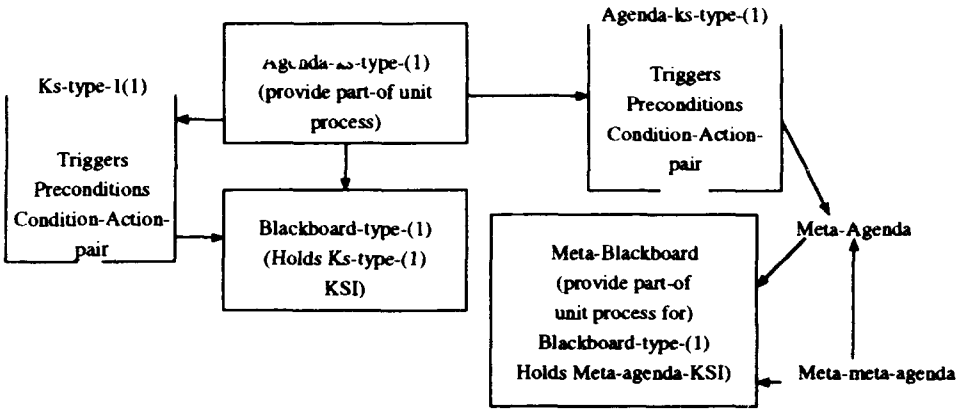


Figure 1

The ks-type-1 can have a number of problem dependent kS attached. To provide solution islands the system may generate ks-type-2 etc. The ks-type can communicate via the agendas and the blackboards. Below shows the structure of a simple bulldozer definition through the strategic model.

If an operation *move* is required specific to the bulldozer then the information to move has to be aquired. The information can be established either from an event or a KS. If an event or KS does not exist for an operation then the structure needs to be built. This system adaption is controlled from the agenda level. Once the possible KS solution has been generated control is passed back to the ***ks-type***. Part of the reflective control occurs through the ***id ks-type***. As all the conditions have an ***id*** they all have the control option.

```
ks-type-1(bulldozer)
     :triggers (((($EVENT-IS 'setup) level(operation))
                 (or $EVENT-IS 'active))
     :precondition (type level active)
     :action (rules:new-rule :id ks-type(bulldozer ret-op)
                     :conditions ($priority-set return-operations)
                     :action
                         (PROPOSE :change-type ($ret-val ret-op)
                                  :entry ($ret-entry ret-op)
                                  :level ($case-rule ret-op)
                                  :attributes ($res-op ret-op)
                                  :value ($res-val ret-op))))
          :local variable-returns))
```

The figure 2 shows how the links are provide and the indexing is achieved for the strategic model.The structure can encompass each component allowing the meta-control look at all of the blackboard. Internally, this view can be the whole structure, that is the meta-control which is also operates as part-of process.

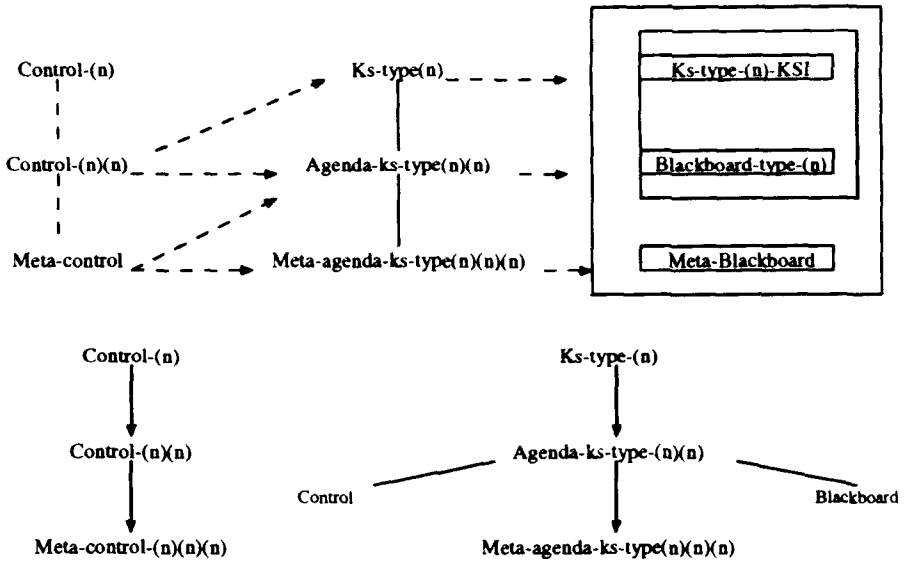


Figure 2

Like OPM we can view parts of the blackboard, which is itself a blackboard, and we can request restructuring from a high level. The network perspective is important at this point of analysis and assists in answering some of the classification problems in abstraction analysis. The overall model allows: **reasoning links**, **partial blackboard analysis**, **updating and the recognition of changes**, **the blackboard to remain distributive**, **the control to remain distributive**, **levels of interpretation from the model to the execution (KSI)**, and **static and dynamic updating from execution which allows an adaptive learning system**.

The network control structure builds the KS from the condition set. This organisation of the conditions depends on how the KS is to interact with the overall problem. The process used to is establish possible linked relationships. Strategies are dependent in their most abstract definition for their approximation of a problem. The reasoning has to be supported for explanation which can be internal to the unit process, part-of the unit, or as whole system. The infinite model is represented by a reflective reasoning process. Knowledge sources are formed from the basic production system to the most complicated meta-agenda.

3.3 Defining the strategy

In most systems a protocol is defined for how a problem can be expressed. Difficulties arise with this when a problem is not well understood or when more information is needed. Each situation is a symptom of the other. Tutoring systems provide environments of increasing [1] information when a student is finding difficulty with a problem. In a problem solving environment the information may be available to solve the problem but the understanding of the information is not

present in a form which can be easily reasoned about. The process of solving a problem can be presented with a solution process. It is this solution process that describes a structure from each process, which may also offer further problem representation in context.

This idea, in itself, is not new: a similar process is used in Case-based reasoning [9, 7] (CBR) systems. Learning by analogy is a method to improve and solve similar problem situations. The analogy method of problem solving uses already solved cases as a domain comparison. The case is modified and or extended to solve the new problem. There is a bootstrap problem with this form of problem solving when there is no case to do a comparison on. Although in reality there is always some level of *case rule* to compare to. A context of abstraction-level is required to employ the right level of *case rule*. The context requires not only the rules but also the appreciation of how and when to use the rule. This form of context analysis can take many forms, for example, to establish a simple condition as true or false or to initialise the strategic outline from the system rules.

In most planning system results are presented in one form only. However, to appreciate the process of problem solving in an abstract fashion, this process cannot be fixed, an open structure is required. This open structure requires a condition analysis process. The first step we take in bringing this system into focus to be implemented is the use of the condition set.

3.4 Defining the Condition Set

Conditions are built by the system state which will have degrees of dynamic interaction. This is determined by the priority setting which decides critical factors from the user event definition. In defining the condition the context of the abstraction definition becomes such that the reflection looks at how the condition reacts to the system state.

Condition-n, as a unit is the statement that represents n for example fire or refuel. To arrive at a condition there must be a before-action (and before the before-action a condition etc): this describes a causal relationship. The usual condition-action pair definition results but the classification of an action is a condition. This allows the initial definition of the world we are interested in to be statements (similar to horn clauses in Prolog [13]). To start our initial reflective condition we look for the most basic element, which in this problem of definition, is the condition or the action. This method is used to provide the constructs for building the more complex structure. The building of the structure is the strategy. By using such a method we are able to reason and present context in the unit form and as a part-of form. This technique of using reflection is not unlike the method used in ELEKTRA [4] when interfacing to the blackboard structure.

The ELEKTRA system is constructed in such a way it supports reflective processing: that is, ELEKTRA programs (sets of production rules) can reason about themselves in a theory-relative, causally-connected fashion. In terms of production systems, this means that production rules are allowed to match other rules and to inspect and reason about their contents. The execution of some

rules, in particular those rules which assigned the status of meta-rules, causes changes to the system which are directly felt at the object level. This goes beyond the normal property of production systems by which the execution (or 'firing') of one rule alters working memory contents, thus enabling some other rules, while disabling others, even though this is one of the main ways in which the causal connection between levels is achieved. In our strategic model we carry this process through to the condition level. This allows conditions to be complete knowledge source structures. Although the condition from one level may be only an expression which returns a value or a structure which requires further evaluation.

As the strategic reflective process is already built before passing the condition to the blackboard, then many of the abstraction problems are resolved. The condition definitions can become part of the blackboard to allow further consistent control to follow through from one system to another.

3.5 Adding control through Reflection

Using the agenda and the knowledge source structure we develop from O-Plan2, NONLIN [14] and OPM ideas incorporating reflection and abstraction. Planning systems demand a form of problem abstraction.

We use layered agendas; these define the KS network and are themselves structured as KS. The agendas are more abstract, but are still knowledge sources which are able to reason. One of the problems outlined by Tate [15] is the choice between modular fine grain KS and efficient large KS. The agenda in KS representation allows KS clusters for a defined point (sub-task). The agendas (Higher-Level KSs) are controlled by a further level, the scheduler. The structure is easy to visualise in a static mode or a snap shot of the "state". However, in a reflective system the control of each component is independent and at the same time has complete control over the system. A heterarchical structure in OPM is used [8] for a further abstraction plane(s). The blackboard model is restrictive in allowing KSs to be aware of other knowledge sources other than via the database with the control through the agenda. In a reflective system all structures have control and are themselves controlled. The hierarchy of a problem is not always a true structure of the problem but it is a way of representing a problem solution. In a strategic system this can be analysed in a number of ways: as a **`new problem definition`**, **`solution`**, **`addition to the system`**, **`control`**, **`success`** or **`failure`**.

The user event meta-system creates events when required either internally or by external request: this is part of the reflective process. This allows a (sub)system to recognise a need and create a "system solution". This will use the building blocks to present what it can contribute to the (sub)system drawing all other possible KSs into action.

As explained earlier the agenda can be posted as a whole structure. A meta structure is used to view the system for a global self-analysis although this will be organised into groups. The control method uses a NONLIN [14] characteristic, that is to allow groups (and sub groups etc) to interrogate structures and create

further analysis structures which contains a context value. The context value is relative to the level of abstraction. This method is used to develop more efficient solutions where efficiency is calculated by the number of steps required to achieve a goal.

Below gives the initial outline structure for the agenda template. The *$name* functions are built from the condition class. This class inherits structure from the trigger language provided in the blackboard interpreter. Once the initial structure is provided by the system the strategic network can be used to put the *condition-set in place*. The whole process may take many passes to provide the solution framework.

```
(declare-user-events 'agenda :new)

(setq ks-agenda
   (def-ks
       :name 'ks-agenda
       :trigger '(or ($agenda-setup-is 'set-up)
                     ($improve-agenda-is 'make-new))
       :precondition '($relationship-set 'agenda)
       :action (list (rules:new-rule :id 'agenda
                                     :conditions t
                                     :action
                                     (PROPOSE :change-type 'make-sub-ks
                                              :entry ($create-agenda)
                                              :level ($create-level)
                                              :attribute ($event-contribution)
                                              :value ($set-macro-type))))
       :local-variable-names '())
```

The agenda is structured as a knowledge source which means that the value context can be represented as a knowledge source. This context representation can take a range: from part of to a complete KS. This is all part of the abstraction and reflection processes. Each level is indexed and can be searched and identified. The condition-action control shows the developing abstraction structure. This differs from other systems as the reasoning mechanism allows the structure to be compartmentalised in a form that can be evaluated as either a unit process or part-of unit process. We are required to make structures explicit for the problem definition representation which requires knowing how to represent structures, when and understanding why.

4 Conclusion

Our organisational method is a reflective model which allows plan repair, provides decision explanation, and new rules to be added. To establish these attributes along with problem context and definition the model needs to view itself and the problem objevtively.

Abstractions even in simple problems can be difficult to define and may not provide a hierarchy: this structure allows the problem definition to move between forms.

To provide blackboard unification the agendas are made up of KS which may be subagendas. Like other planners, it reduces complexity by providing another level of control. As part of a control method the system can force a value context, this can only occur once parts of the problem have been well defined through the system structure. For parts of the problem to become well defined and for solutions to form may take many passes through the system. Our model provides mechanisms to: allow statement definition through the condition set, use the natural expert system strategy for solution expression, overcome the blackboard interpreter problems, improve abstraction definition in problems, allows strategic analysis, and provides reflection for an adaptive learning environment with control.

Our model and blackboard system are both being developed in Common Lisp. We would like to thank Iain Craig who has allowed us to use his blackboard interpreter for the basis of our research.

References

1. D. Bierman, J and P. Kamsteeg, A. Elicitation of knowledge with and for *intelligent* tutoring systems. *International Journal of Educational Research*, pages 799–807, 1988.
2. R. Cohen, D. Greenberg, D. Hart, and A. Howe. Trial by fire. *AI Magazine*, 10(3):33–48, fall 1989.
3. I. D. Craig. The blackboard architecture: A definition and its implications. Research report, University of Warwick, 1987.
4. I. D. Craig. ELECTRA: A Reflective Production System. Research report, University of Warwick, 1991.
5. R. Englemore and T. Morgan. *Blackboard Systems*. Addison and Wesley, 1988.
6. Giunchiglia. A system for Multi-level Mathematical Reasoning. *Artificial Intelligence in Mathematics*, 1991.
7. K. J. Hammond. CHEF: A Model of Case-based Planning. *Readings in Planning*, 1990.
8. F. Hayes-Roth, B. Hayes-Roth. A cognitive model of planning. *Readings in Planning*, pages 245–262, 1990.
9. B. Lopez and E. Plaza. Case-based Learning of Strategic Knowledge. *European Working Session on Learning:LN in AI*, 1991.
10. E. Plaza. Inference-level reflection in NOOS. *IMSA International Workshop on Reflection and Meta-level Architectures*, 1992.
11. J. Rice. The Design and Implementation of *poligon*, a High-Performance, Concurrent Blackboard System Shell. report STAN-CS-89-1294, Stanford University November 1989.
12. B. Smith. Reflection and semantics in lisp. *ACM*, pages 23–35, 1983.
13. L. Sterling and E. Shapiro. *The Art of Prolog*. MIT press, 1986.
14. A. Tate. Generating Project Networks. *Readings in Planning*, 1990.
15. A. Tate. O-Plan2: Choice Ordering Mechanisms In an AI Planning Architecture", Journal = "Worhshop on innovative approaches to planning, scheduling and control. 1990.
16. D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Readings in Planning*, 1990.

On the Learning of Rule Uncertainties and their Integration into Probabilistic Knowledge Bases

Beat Wüthrich
ECRC GmbH, Arabellastr. 17
D-8000 Munich 81, Germany
e-mail: beat@ecrc.de

Abstract. We present a natural and realistic knowledge acquisition and processing scenario. In the first phase a domain expert identifies deduction rules where he thinks that they are good indicators of a specific target concept to occur. Then, in a second knowledge acquisition phase, a learning algorithm automatically adjusts, corrects and optimizes the deterministic rule hypothesis given by the domain expert by selecting an appropriate subset of the rule hypothesis and by attaching uncertainties to them. Finally, in the running phase of the knowledge base we can arbitrarily combine the learned uncertainties of the rules with uncertain factual information.

1 Introduction

The aim of this study is twofold. First, we show how the learning techniques for propositional concepts given by [KS90] can be used to learn probabilities of not necessarily propositional deduction rules. Second, we show how to integrate or simulate these learned probabilities in a rule-based calculus which deals quantitatively with vague rule premises [Wüt92b]. Together, this gives a knowledge representation tool for uncertain rule-based reasoning.

We start by saying why we think that a combination of rule-based reasoning and probabilities is a promising and needed approach to capture the knowledge of domain experts into a knowledge base system. Humans like to think in terms of disjunctions of conjunction (e.g. see [Val85, p.560]) or in terms of rules like

$$\mathit{has_cancer}(x) \leftarrow \mathit{person}(x) \wedge \mathit{smoker}(x) \quad (1)$$

$$\mathit{has_cancer}(x) \leftarrow \mathit{person}(x) \wedge \neg \mathit{does_sport}(x) \quad (2)$$

$$\mathit{has_cancer}(x) \leftarrow \mathit{person}(x) \wedge \mathit{ancestor}(y, x) \wedge \mathit{has_cancer}(y) \quad (3)$$

However, in many situations to get from a domain expert such rules is not the end of the knowledge acquisition task since these rules do not express precisely enough the reality. For instance, the rules (1),(2) and (3) are only true to a certain degree. This raises the need to have a means to deal with "degreeness" of truth. There are two sources of vagueness. The rule (1) itself holds only to a certain degree, and, the premise $\mathit{smoker}(x)$ needed to deduce the conclusion $\mathit{has_cancer}(x)$ can also be more or less true. For instance, $\mathit{smoker}(\mathit{hans}) : 0$, $\mathit{smoker}(\mathit{hans}) : 0.5$, and $\mathit{smoker}(\mathit{hans}) : 1$ can model respectively that *hans* is a non-smoker, smokes one package of cigarettes a day and smokes more than

two packages a day. So in many situations the rule-based paradigm alone is not sufficient. We need a way to express uncertainty. Also, a combination of rules and uncertainties can be used to simulate default reasoning in rule-based systems [Ric83]. Note that an uncertainty can be interpreted in different ways. The fact $smoker(hans) : 0.5$ can either express that *hans* is a medium strong smoker or, that he is a smoker or a non-smoker but we have absolutely no hint which of the two possibilities is the right one.

There are basically two ways of dealing with vagueness, either *quantitatively* or *qualitatively*. We will adopt the former one and try to formalize and handle the knowledge quantitatively. However, quantitative methods have a serious drawback versus qualitative approaches like the one proposed in [HR83]. As stated there, the main problem with probabilistic reasoning is that people are unwilling to give precise numerical probabilities to outcomes. Therefore, Halpern and Rabin proposed in the former study a qualitative, propositional logic of likelihood to deal with the intuitively appealing notion of "likely" without using explicit numbers or probabilities. Since people are unwilling to give precise values to outcomes, it is exceedingly important to show how to eliminate the need that people have to estimate probabilities of deduction rules; especially when the task is also complicated by the fact that one has to give numerical values for the eight different cases that $has_cancer(x)$ holds if only the rule body of (1) is true, if exactly the bodies of (1) and (2) are true, and so on. While the heuristic estimation of probabilities of rules seems indeed to overtax the human capabilities, an appropriate setting of the truth degree of individual facts or ground atoms seems to be manageable by domain experts. For instance, to attach a number to $exercises(hans, soccer)$ expressing the degree of truth for this fact is not more or less difficult than to judge whether $exercises(hans, soccer)$ necessarily holds, does not necessarily hold, likely holds, possibly holds and so on - to speak in terms of the language introduced in [HR83].

We will overcome the problem of probability estimation by giving a learning algorithm for probabilities of rules. Furthermore we will show how these numbers can be dealt with in a rule-based system also dealing with uncertain premises. Together this results in a framework where people can think in terms of (not necessarily propositional) deduction rules and where the reasoning is more precise than in the framework proposed in [HR83]. Our framework allows preciser information because event dependencies cannot only be "likely" or "conceivable" as in [HR83] - p occurs likely or conceivablely respectively if q occurs - but $p \leftarrow q : \gamma$ can express any degree of dependence between "p holds" if "q holds". The degree of dependence is expressed by $\gamma \in [0, 1]$. The use of explicit numbers circumvents also a serious problem occurring in Halpern's and Rabin's likelihood logic [HR83, p.318]. Let us interpret "likely" as being greater or equal to 0.5. Now consider a situation where we toss a fair coin twice, and let p represent "the coin lands heads both times", while q represents "the coin lands tails both times". Then from the likelihood logic follows that since $p \vee q$ is likely

either p is likely or q is likely, which is not the case. On the other hand, using our approach this problem is avoided. In terms of rules and facts, the former situation is $p \leftarrow head(1) \wedge head(2)$, $q \leftarrow \neg head(1) \wedge \neg head(2)$, $head(1) : 0.5$ and $head(2) : 0.5$. Then using the rule-base calculus proposed in [Wüt92b] we get the probability of event p and those for event q to be 0.25, the probability of $p \vee q$ to be 0.5, the probability of both events occurring, i.e. $p \wedge q$, to be zero, and the probability of $\neg q \vee p, p \vee \neg q, \neg q$ and also $\neg p$ to be 0.75.

Now we come to the delicate task of making some realistic assumptions on the real world. Suppose a domain expert identifies a set of relevant rules, like (1) to (3) in order to define when a concept like $has_cancer(x)$ occurs for a particular person x. Then we distinguish eight cases: either none of the three rule bodies is true for this person, or only the rule body of (1) is true, ..., or only the rule bodies of (1) and (2) are true, or all three rule bodies are true. In either case, there is a specific probability for this person having cancer. Hence, in this example, we assume that there are probabilities $\gamma_1, \gamma_2, \gamma_3, \gamma_{12}, \gamma_{13}, \gamma_{23}, \gamma_{123}$ satisfying the following. Whenever we investigate in future a randomly drawn person like *hans* where all premises like $execises(hans, soccer)$ or $smoker(hans)$ are either true or false then $has_cancer(hans)$ has probability 0 if non of the rule bodies of (1) to (3) *is true wrt the substitution* $\{x/hans\}$; it has probability γ_1 if exactly rule body (1) is true wrt $\{x/hans\}$; it has probability γ_{23} if precisely the body of (2) and (3) is true wrt $\{x/hans\}$; and so on. When saying that for instance the body of (3) is true wrt $\{x/hans\}$ we mean that the existential closure of $(3)\{x/hans\}$ is true, i.e. that $\exists y(person(hans) \wedge ancestor(y, hans) \wedge has_cancer(y))$ is true. Note that the assumption of the existence of the numbers $\gamma_{\vec{i}}$ seems indeed to be realistic since if we add the additional rule $has_cancer(x) \leftarrow person(x)$ then we will also capture those persons having cancer but for which none of the rule bodies is true. However, in this case we then assume the existence of the fifteen numbers $\gamma_1, ...\gamma_{1234}$ with their straightforward meaning.

2 Preliminaries

We define *the model of efficient distribution-free learning of probabilistic concepts* as it was introduced in [KS90]. This model is a natural and important extension to Valiant's probably approximately correct learning model for (deterministic) concepts [Val84, Nat91].

A *probabilistic concept*, abbreviated by p-concept, over a domain set (or *instance space*) X is simply a mapping $c : X \rightarrow [0, 1]$. For each $a \in X$, we interpret $c(a)$ as the probability that a is a positive example of the p-concept c. A learning algorithm is attempting to infer something about the underlying target p-concept c solely on the basis of labeled examples (a, b), where $b \in \{0, 1\}$ is a bit generated randomly according to the conditional probability $c(a)$. Thus, examples are of the form $(a, 0)$ or $(a, 1)$ - not $(a, c(a))$.

A *p-concept class* C is a family of p-concepts. On any execution, a learning

algorithm is attempting to learn a distinguished *target* p-concept $c \in C$ with respect to a fixed but unknown and arbitrary target distribution D over X. The learning algorithm is given access to an oracle EX that behaves as follows: EX first draws a point $a \in X$ randomly according to the distribution D. Then with probability $c(a)$, EX returns the labeled example $(a, 1)$ and with probability $1 - c(a)$ it returns $(a, 0)$.

Now in this setting we are interested in inferring a good model of probability with respect to the target distribution. We say that a p-concept h is an *ϵ-good model of probability* of c with respect to D if we have $Pr_{a \in D}[\mid h(a) - c(a) \mid \leq \epsilon] \geq 1 - \epsilon$. Hence the value of h must be near that of c on almost all points a.

Let C be a p-concept class over domain X. We say that C is *learnable with a model of probability* if there is an algorithm A such that for any target p-concept $c \in C$, for any target distribution D over X, for any inputs $\epsilon, \delta > 0$, algorithm A, given access to EX, halts and with probability $1 - \delta$ outputs a p-concept h that is an ϵ-good model of probability for c with respect to D. We say that C is *polynomially learnable* if A runs in time $poly(1/\delta, 1/\epsilon)$, i.e. in time polynomial in $1/\epsilon$ and $1/\delta$.

We need an additional assumption in order to be able to use Kearn's and Schapire's learning model also for non-propositional concepts. Assume to have some rule bodies $d_1(\overline{x}), ..., d_m(\overline{x})$. Then oracle EX returns not only a randomly drawn example $\overline{a} \in X_1 \times ... \times X_n$ for some $n > 0$ together with the bit $b \in \{0, 1\}$, but the oracle will also tell us whether $d_i(\overline{a})$ for $1 \leq i \leq m$ is true or not. The closed formula $d_i(\overline{a})$, $\overline{a} = a_1, ..., a_n$, is obtained from $d_1(\overline{x})$, $\overline{x} = x_1, ..., x_n$, by applying the ground substitution $\{x_1/a_1, ..., x_n/a_n\}$. Thus, the oracle EX returns the labeled example $(\overline{a}, b_1, ..., b_m, b)$, where $b_1, ..., b_m, b \in \{1, 0\}$. This extension implies that if a domain expert gives an indicator $d_i(\overline{x})$ then someone (the oracle or a set of background information in the form of facts and rules respectively) has to be able to tell us whether $d_i(\overline{a})$ is true or not for some point $\overline{a} \in X_1 \times ... \times X_n$. So if we draw for instance *hans* during the *learning phase* the oracle decides whether *hans* is a smoker or not. However, this does not imply that later on when having learned the concept we may not value persons of having cancer where we are not one hundred percent sure whether these persons are now non-smokers or smokers. Or in other words, although we will cope later on with uncertain premises, we assume certain premises during the learning phase. As we will see, the truth value of the premises is not estimated directly but indirectly giving numbers to ground atoms only. The assumption that the oracle has to decide during the learning phase whether a premise is true or not can be justified as follows. Whenever we have a training example and we (the oracle or background information in form of rules and facts respectively) can not decide on the truth of the premises then we simply skip this example and try another one. This implies that if we would need $poly(1/\delta, 1/\epsilon)$ training examples to learn a particular concept and if we can decide only on β of them whether the premises are satisfied or not then we actually have to draw $poly(1/\delta, 1/\epsilon)/\beta$

training examples. Consequently, we will learn the uncertainties of the rules on a subset X' of the domain X and assume that the examples from X' behave as the examples from X. So we extrapolate from X' to X. But for the mathematical analysis of the learning algorithm, we will assume that the trainings examples are randomly drawn from X according to the unknown and arbitrary distribution D. Note that we do not allow an additional length parameter like the number of relevant indicators (e.g. see [Nat91, section 2.1]). The reason is that a domain expert can realistically be asked to give at most m relevant indicators for some fixed m. Moreover, if we would allow this additional complexity parameter then the class of concepts we are particularly interested in here (vdp-concepts defined below) would neither be efficiently learnable nor could these concepts be evaluated in polynomial time. Hence, we are really forced to impose an upper bound on the number of potentially relevant indicators.

We discuss syntactic restrictions on the rule bodies. We tacitly assume that the given rule bodies $d_1(\overline{x}), ..., d_m(\overline{x})$ are function-free, conjunctions of literals, all having the same free variables and that the domain $X_1 \times ... \times X_n$ from which the oracle will randomly draw the values for $\overline{x} = x_1, ..., x_n$ is explicitly encoded in each rule body in the form $dom(x_1) \wedge ... \wedge dom(x_n)$, i.e. in the form of an additional conjunction of atoms. In rule (1) to (3) this conjunction is just $person(x)$. We also impose that the rule bodies are *allowed* (each head variable occurs in a positive body literal). This has no consequence on the learning itself but ensures good properties when afterwards combining the learned probabilities with a special rule-based calculus.

In this study we are particularly interested in the class C of *visible disjunctive p-concepts* with respect to $\{d'_1(\overline{x}), ..., d'_m(\overline{x})\}$ - abbreviated by *vdp-concepts* (wrt $\{d'_1(\overline{x}), ..., d'_m(\overline{x})\}$). When appropriate we write d_i instead of $d_i(\overline{x})$. Each rule body d'_i obeys the previously defined restrictions. A vdp-concept $c \in C$ wrt $\{d'_1(\overline{x}), ..., d'_m(\overline{x})\}$ is defined by a set of n rule bodies $\{d_1, ..., d_n\} \subseteq \{d'_1, ..., d'_m\}$ and by $2^n - 1$ real numbers γ_{i_1} $(1 \leq i_1 \leq n), \gamma_{i_1 i_2}$ $(1 \leq i_1 < i_2 \leq n), \gamma_{i_1 i_2 i_3}$, $(1 \leq i_1 < i_2 < i_3 \leq n), ...\gamma_{i_1 i_2 ... i_n}$, $(1 \leq i_1 < i_2 ... < i_n \leq n)$. Each real value $\gamma_{\overline{i}}$ is in the interval $[0, 1]$. We use the convention $\gamma_{\overline{i}} = \gamma_{\overline{i}'}$ if $\overline{i}'$ is a permutation of the indices $\overline{i}$. For a point $\overline{a} \in X$, we have $c(\overline{a}) = \gamma_{12...k}$, where $\{d_1(\overline{a}), d_2(\overline{a}), ..., d_k(\overline{a})\}$ is the maximal subset of $\{d_1(\overline{a}), ..., d_n(\overline{a})\}$ such that each $d_i(\overline{a})$ for $1 \leq i \leq k$ is true. Otherwise, if no disjunct is true wrt $\overline{a}$ then $c(\overline{a}) = 0$.

3 Learning Rule Uncertainties

Proposition 1 *Visible disjunctive probabilistic concepts wrt $\{d'_1, ..., d'_m\}$ are polynomially learnable.*

Proof. The learning algorithm taking as input δ and ϵ makes direct use of a technique suggested in [KS90, p.386] and performs in two steps. First, for each of the 2^m possible hypothesis $h = \{d_1, ..., d_n\}$ wrt $\{d'_1, ..., d'_m\}$ we do the following.

We estimate $\gamma_{1...l}$ as the frequency with which the target concept occurs in a randomly drawn new sample of size $poly(1/\delta, 1/\epsilon)$ if $d_1 \wedge ... \wedge d_l \wedge \neg d_{l+1} \wedge ... \wedge \neg d_n$ is true. If this disjunct is never true then we can set $\gamma_{1...l}$ to any real number in $[0, 1]$. This yields an estimation $\hat{h}$ for each of the 2^m hypothesis h. Second, for each of the 2^m hypothesis $\hat{h}$ we estimate its error using the quadratic loss method proposed in [KS90, p.386] and choose the best possible hypothesis. This is done using the function $Q_{\hat{h}}$ for a hypothesis $\hat{h}$ which is specified for an example (x, y) as $Q_{\hat{h}}(x, y) = (\hat{h}(x) - y)^2$. For each individual $\hat{h}$ we draw $poly(1/\delta, 1/\epsilon)$ many new examples $(x_1, y_1), ...(x_n, y_n)$ randomly from X and estimate with this sample the expected loss $E[Q_{\hat{h}}]$ as $\hat{E}[Q_{\hat{h}}] = 1/n * (Q_{\hat{h}}(x_1, y_1) + ... + Q_{\hat{h}}(x_n, y_n))$. Then we output as ϵ-good model of probability a hypothesis $\hat{h}$ with minimal empirical loss $\hat{E}[Q_{\hat{h}}]$. The prove that the algorithm performs as claimed is given in [Wüt92a]. $\diamond$

4 Integrating vdp-Concepts

We show how a learned vdp-concept can be handled by the rule-base calculus [Wüt92b]. We briefly introduce the main characteristics of this calculus first. Let $KB = (F, R)$ be a knowledge base consisting of a finite set of facts or ground atoms F, each having attached a probability in the closed real interval between zero and one, and a finite set of stratified deduction rules R. Then the mentioned calculus computes for each ground formula a probability which reflects whether it is true or false with respect to the given facts and rules. For example, on the rules

$$\begin{aligned} has_cancer(x) &\leftarrow person(x) \wedge ancestor(y, x) \wedge has_cancer(y) \\ ancestor(x, y) &\leftarrow parent(x, y) \\ ancestor(x, y) &\leftarrow parent(x, z), ancestor(z, y) \end{aligned}$$

and the facts $\{parent(d, c) : 1, parent(c, b) : 0.8, has_cancer(d) : 0.7, person(d), person(c), person(b)$ the calculus computes for $has_cancer(b)$ a probability of 0.56, denoted $\mathcal{I}_{KB}(has_cancer(b)) = 0.56$, and for $ancestor(d, b)$ a probability of 0.8. Note that if for some information there is not given an explicit uncertainty, like for $person(d)$ or the rules, then this information is taken to be sure. If we interpret each ground atom as denoting a special ground formula (under the minimal model semantics of [ABW88], $has_cancer(b)$ denotes $person(b) \wedge parent(d, c) \wedge parent(c, b) \wedge has_cancer(d)$ in our example) then for this calculus the following properties can be shown:

1. $\mathcal{I}_{KB}(p) \leq \mathcal{I}_{KB}(q)$ if p logically implies q.
2. $\mathcal{I}_{KB}(p) \geq 0$
3. $\mathcal{I}_{KB}(p \vee q) = \mathcal{I}_{KB}(p) + \mathcal{I}_{KB}(q)$ if p and q are mutually exclusive i.e. $p \wedge q$ is unsatisfiable.

4. $\mathcal{I}_{KB}(p) = 1$ if p is the certain event, i.e. a tautology.

Thus, on the boolean algebra $([S], \wedge, \vee)$, where S is a set of ground formulas and $[S]$ is S modulo logical equivalence, we have that the mapping $\mathcal{I}_{KB}$ is well defined, i.e. $\mathcal{I}_{KB}(p) = \mathcal{I}_{KB}(q)$ if $p \leftrightarrow q$, and that $\mathcal{I}_{KB}$ defines a probability function in the sense of axiomatic probability theory. Thus, all the usual consequences of probability theory also hold here [Par60, pp.18-22] when replacing the relation $\subseteq$ between events by $\rightarrow$. Moreover, from property 1 we have that the calculus is monoton in the number of derivations for a particular ground atom. For instance, if $has_cancer(b)$ could also be deduced by $person(b) \wedge smokes(b)$ then the probability of $has_cancer(b)$ increases. In [Wüt92b] is also shown how to approximate the function $\mathcal{I}_{KB}$ in time polynomial in the number of the facts F and that this calculus reduces to the usual minimal model $M_{F \cup R}$ of the stratified Datalog program $F \cup R$ (see [ABW88]) if each fact in F is sure.

We now outline how to simulate a learned vdp-concept c in the calculus introduced. For a vdp-concept defined by n rules we solve a system of $2^n - 1$ linear equalities in $2^n - 1$ unknowns $x_{\vec{i}}$. In our example, we solve

$$
\begin{aligned}
\gamma_1 &= x_1 * x_{12} * x_{13} * x_{123} \\
\gamma_2 &= x_2 * x_{12} * x_{23} * x_{123} \\
\gamma_3 &= x_3 * x_{13} * x_{23} * x_{123} \\
\gamma_{12} &= \gamma_1 + \gamma_2 - x_1 * x_2 * x_{12} * x_{23} * x_{13} * x_{123} \\
\gamma_{13} &= \gamma_1 + \gamma_3 - x_1 * x_3 * x_{12} * x_{23} * x_{13} * x_{123} \\
\gamma_{23} &= \gamma_2 + \gamma_3 - x_2 * x_3 * x_{12} * x_{23} * x_{13} * x_{123} \\
\gamma_{123} &= x_1 * x_{12} * x_{13} * x_{123} + x_2 * x_{12} * x_{23} * x_{123} + x_3 * x_{13} * x_{23} * x_{123} - \\
&\quad x_1 * x_2 * x_{12} * x_{23} * x_{13} * x_{123} - x_1 * x_3 * x_{12} * x_{23} * x_{13} * x_{123} - \\
&\quad x_2 * x_3 * x_{12} * x_{23} * x_{13} * x_{123} + x_1 * x_2 * x_3 * x_{12} * x_{23} * x_{13} * x_{123}
\end{aligned}
$$

Note that if $\gamma_{\vec{i}} = 0$ then we approximate this by setting $\gamma_{\vec{i}}$ to an arbitrary small but non zero value. These equations have exactly one solution provided they are *friendly defined*, i.e. no divisor happens to be zero and non of the $\gamma_{\vec{i}}$ is zero. But clearly also the case that a divisor is zero can be approximated arbitrarily precise by replacing it by any value greater than zero. Now assume that these equations have been solved yielding in our example $x_1 = \frac{\gamma_1+\gamma_2+\gamma_3-\gamma_{12}-\gamma_{13}-\gamma_{23}+\gamma_{123}}{\gamma_2+\gamma_3-\gamma_{23}}$, $x_{23} = \frac{(\gamma_1+\gamma_3-\gamma_{13})*(\gamma_1+\gamma_2-\gamma_{12})}{\gamma_1*(\gamma_1+\gamma_2+\gamma_3-\gamma_{12}-\gamma_{13}-\gamma_{23}+\gamma_{123})}$, $x_{123} = \frac{\gamma_1*\gamma_2*\gamma_3*(\gamma_1+\gamma_2+\gamma_3-\gamma_{12}-\gamma_{13}-\gamma_{23}+\gamma_{123})}{(\gamma_1+\gamma_2-\gamma_{12})*(\gamma_1+\gamma_3-\gamma_{13})*(\gamma_2+\gamma_3-\gamma_{23})}$ and so on, if each left hand side is non zero. Then we generate the rules

$$
\begin{aligned}
has_cancer(x) &\leftarrow per(x) \wedge smoker(x) \wedge a_1 \wedge a_{12} \wedge a_{13} \wedge a_{123} \\
has_cancer(x) &\leftarrow per(x) \wedge \neg does_sport(x) \wedge a_2 \wedge a_{12} \wedge a_{23} \wedge a_{123} \\
has_cancer(x) &\leftarrow per(x) \wedge anc(y, x) \wedge has_cancer(y) \wedge a_3 \wedge a_{13} \wedge a_{23} \wedge a_{123}
\end{aligned}
$$

and add the new facts $\{a_1 : x_1, a_2 : x_2, a_3 : x_3, a_{12} : x_{12}, a_{13} : x_{13}, a_{23} : x_{23}, a_{123} : x_{123}\}$. We will here not write down how an arbitrary vdp-concept is simulated

but this as well as the resulting values for the $x_{\vec{i}}$ are easily generalized from the above example. Of more interest is the question whether the presented simulation technique is correct or not (see [Wüt92a] for the proofs of proposition 2 and 3).

Proposition 2 *(correctness of the simulation of vdp-concepts for certain rule premises) Let c be a vdp-concept defined by $\{d_1, ..., d_n\}$ and $\gamma_1, ..., \gamma_{1...n}$, such that the rules $R = \{c \leftarrow d_1, ..., c \leftarrow d_n\}$ are stratified, and the corresponding system of $2^n - 1$ equalities for the $x_{\vec{i}}$ is friendly defined. Let R' be the set of rules $\{c \leftarrow d_1 \wedge a_1 ... \wedge a_{1...n}, ..., c \leftarrow d_n \wedge a_n \wedge a_{1...n}\}$, F' be the set of facts $\{a_1 : x_1, ..., a_{1...n} : x_{1...n}\}$ and I be a Herbrand interpretation (which can be regarded as a set of facts each with attached probability one) defining whether a disjunct d_i is true or not wrt an example $\overline{a}$. Then for each point $\overline{a}$ and each interpretation I, if R is non-recursive or $I = M_{I \cup R} - \{c(\overline{a})\}$, then $\mathcal{I}_{(I \cup F', R')}(c(\overline{a})) = \gamma_{1...k}$, where $\{d_1(\overline{a}), ..., d_k(\overline{a})\}$ is the maximal subset of $\{d_1(\overline{a}), ..., d_n(\overline{a})\}$ which is true wrt I.*

5 Combining Rule and Premise Uncertainties

We show now what happens if a vdp-concept is evaluated under uncertain premises. Let us take the rule

$$has_cancer(x) \leftarrow person(x) \wedge ancestor(y, x) \wedge has_cancer(y) \qquad (4)$$

where we know that it is true in 0.1 percent of the cases. Furthermore, we have five fact sets

$$\begin{aligned}
F_1 &= \{ancestor(fritz, hans), ancestor(heiri, hans), \\
&\quad has_cancer(fritz) : 0.8, has_cancer(heiri) : 0.3\} \\
F_2 &= \{ancestor(fritz, hans), has_cancer(fritz) : 0.8\} \\
F_3 &= \{ancestor(heiri, hans), has_cancer(heiri) : 0.2\} \\
F_4 &= \{ancestor(fritz, hans), ancestor(heiri, hans), has_cancer(fritz) : 0.8\} \\
F_5 &= \{ancestor(heiri, hans), ancestor(fritz, hans), has_cancer(heiri) : 0.2\}
\end{aligned}$$

Using our concepts introduced we get

$$\begin{aligned}
0.08 &= \mathcal{I}_{(F_2, \{(4)\})}(has_cancer(hans)) = \mathcal{I}_{(F_4, \{(4)\})}(has_cancer(hans)) \\
0.02 &= \mathcal{I}_{(F_3, \{(4)\})}(has_cancer(hans)) = \mathcal{I}_{(F_5, \{(4)\})}(has_cancer(hans)) \\
0.084 &= \mathcal{I}_{(F_1, \{(4)\})}(has_cancer(hans))
\end{aligned}$$

Proposition 3 gives the precise information on what actually happens in the non-recursive case and assures the correctness of the combination of rule uncertainties with vague premises under the assumption that the rule uncertainties are independent from the premise uncertainties. We write $Pr[A]$ instead of $\mathcal{I}_{(F,R)}(A)$ since it is clear to which facts and rules we refer.

Proposition 3 *(correctness of the simulation of non-recursive vdp-concepts under vague premises) Let (F, R) be a knowledge base where predicate c does not occur and let c be a vdp-concept defined by the disjuncts $\{d_1, ..., d_n\}$ and the friendly real numbers $\gamma_1, ..., \gamma_{1...n}$. Then for any point $\overline{a}$, we have:*

$$\begin{aligned} Pr[c(\overline{a})] \;=\; & Pr[d_1(\overline{a}) \wedge \neg d_2(\overline{a}) \wedge ... \wedge \neg d_n(\overline{a})] * \gamma_1 \\ & + Pr[\neg d_1(\overline{a}) \wedge d_2(\overline{a}) \wedge \neg d_3(\overline{a}) \wedge ... \wedge \neg d_n(\overline{a})] * \gamma_2 \\ & + ... \\ & + Pr[\neg d_1(\overline{a}) \wedge d_2(\overline{a}) \wedge \neg d_3(\overline{a}) \wedge d_4(\overline{a}) \wedge \neg d_5(\overline{a}) \wedge ... \wedge \neg d_n(\overline{a})] * \gamma_{24} \\ & + ... \\ & + Pr[d_1(\overline{a}) \wedge ... \wedge d_n(\overline{a})] * \gamma_{1...n} \end{aligned}$$

and therefore $0 \leq Pr[c(\overline{a})] \leq 1$.

The conclusion of proposition 3 can be seen as the formula of the total probability. If $A_1 \vee ... \vee A_n$ is the certain event then, according to probability theory, $Pr[B] = \sum_{i=1}^{n} Pr[A_i] * Pr[B \mid A_i]$ holds. So, since the γ's are the conditional probabilities $Pr[B \mid A_i]$ and since the conditional probability $Pr[c \mid \wedge_i \neg d_i]$ is by definition zero, the stated fact follows.

The presented simulation technique can also be used to express other non-obvious cases. Assume to have three mutually exclusive events b_1, b_2 and b_3 with probability cf_{b_1}, cf_{b_2} and cf_{b_3} respectively. It is now no longer obvious how to express mutual exclusiveness as for example when having *two* mutually exclusive events (see our example "we toss a fair coin twice" given in section 1). We set $p_1 \leftarrow b_1 \neg b_2 \neg b_3 a_1, ..., p_3 \leftarrow \neg b_1 \neg b_2 b_3 a_3$ and attach for instance to a_1 the uncertainty $1/((1 - cf_{b_2}) * (1 - cf_{b_3}))$. This assures the desired effect that $Pr[p_i] = Pr[b_i] = cf_{b_i}$, $Pr[p_i \wedge p_j] = 0$, $Pr[p_i \vee p_j] = Pr[p_i] + Pr[p_j]$ and $Pr[p_1 \vee p_2 \vee p_3] = cf_{b_1} + cf_{b_2} + cf_{b_3}$ for $1 \leq i < j \leq 3$. So we then look at p_i instead of b_i which results in the desired behavior.

6 Conclusions

This study combines results from three different areas: probability theory, rule-based reasoning and machine learning. In particular, we presented a natural and realistic knowledge acquisition and processing scenario. A domain expert gives a couple of hopefully strong indicators of whether a target concept will occur. A learning algorithm then selects a subset of these rules and attaches weights to them such that the concept will be predicted probably optimally within the bounds of the original rules. Although we do not make any assumption on the correctness of the domain expert's specification, it is clear that the better the original rules are the better results the learning algorithm can produce. Finally, we integrate the learned concepts into probabilistic knowledge bases where we

can also give the probability of a concept to occur even when the rule premises are vague. Moreover, different learned concepts and non-learned deterministic rules can be added together yielding a large uniform knowledge base.

We believe that a prerequisite for a successful real world application of uncertain information in knowledge bases is to have available a "good" uncertainty calculus as well as "good" uncertainties themselves. We hope we have satisfied both prerequisites.

References

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1988.

[HR83] J. Y. Halpern and M. O. Rabin. A Logic to Reason About Likelihood. In *Proc. Annual ACM Symp on the Theory of Computing*, pages 310–319, 1983.

[KS90] M. J. Kearns and R. E. Schapire. Efficient Distribution-free Learning of Probabilistic Concepts. In *Proc 31st IEEE Symposium on Foundations of Computer Science*, pages 382–391, 1990.

[Nat91] B. K. Natarajan. *Machine Learning. A Theoretical Approach*. Morgan Kaufmann, 1991.

[Par60] E. Parzen. *Modern Probability Theory and its Applications*. John Wiley & Sons, Inc., 1960.

[Ric83] E. Rich. Default Reasoning as Likelyhood Reasoning. In *Proc. National Conf on Artificial Intelligence (AAAI-83)*, pages 348–351, 1983.

[Val84] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Val85] L. G. Valiant. Learning Disjunctions of Conjunctions. In *Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 560–566, 1985.

[Wüt92a] B. Wüthrich. On the Efficient Distribution-free Learning of Rule Uncertainties and their Integration into Probabilistic Knowledge Bases. Technical Report 92-28, ECRC, 1992.

[Wüt92b] B. Wüthrich. Probabilistic Knowledge Bases. *submitted to the Journal of Logic Programming*, 1992. extension of Towards Probabilistic Knowledge Bases in Springer Verlag, Lecture Notes in AI, Nr 624.

Recognition of Functional Dependencies in Data

Robert Zembowicz and Jan M. Żytkow

Department of Computer Science, Wichita State University, Wichita, KS 67208

Abstract. Discovery of regularities in data involves search in many spaces, for instance in the space of functional expressions. If data do not fit any solution in a particular space, much time could be saved if that space was not searched at all. A test which determines the existence of a solution in a particular space, if available, can prevent unneeded search. We discuss a functionality test, which distinguishes data satisfying the functional dependence definition. The test is general and computationally simple. It permits error in data, limited number of outliers, and background noise. We show, how our functionality test works in database exploration within the 49er system as a trigger for the computationally expensive search in the space of equations. Results of tests show the savings coming from application of the test. Finally, we discuss how the functionality test can be used to recognize multifunctions.

1 Introduction: the Role of Application Conditions

Machine Discovery systems such as BACON [4], FAHRENHEIT [11], KDW [7], EXPLORA [2,3], FORTY-NINER (49er) [9,10], and many others, explore data in search for hidden knowledge. They conduct search in usually large spaces of hypotheses. More discoveries can be made when a hypotheses space is larger, but search in a large space takes plenty of resources: time and memory. Using 49er to explore a database, for instance, even if a single search for an equation may take few seconds, when repeated thousands of times it can consume very significant resources. Can unsuccessful search be avoided? What tests can be applied to determine whether there is a chance for a solution in a particular space?

Another reason to avoid the search is when results would not have meaning. Different types of regularities make sense for different types of variables. For instance, even if two variables have numerical values, if the numbers are on the nominal scale rather than on interval or ratio scales, the discovered equations are not meaningful. Similarly for a nominal attribute it does not make sense to aggregate the values in two or more classes classes to summarize data in simple contingency tables, because in almost all cases the apriori conducted grouping has little sense. Simple conditions, which do not even consider the data, but only domain knowledge about the type of variables, can be sufficient. Zembowicz and Żytkow [9] summarize the dependence between variable type and regularity type for the use of 49er. All *application tests* are evaluated before the corresponding space is searched, so that the number of spurious regularities decreases and/or the efficiency of search is improved.

In this article we will concentrate on the functionality test that prevents an unsuccessful but costly search in the space of functional expressions. Rather than considering only types of variables, our test analyzes the actual data.

2 Testing Functionality

Empirical equations are efficient summaries of data, if the data distribution is close to a functional relationship. In many machine discovery systems, equations form large, potentially unbound search spaces [1,4,5,6,8,9]. The search in the space of equations is expensive, and it may be repeated many times for different combinations of variables and different ranges of data. It would be a good idea to avoid search on the data which are not conducive to functional description. The problem occurs typically in database discovery, where most of data follow weak statistical patterns rather than strong functional dependencies, but still some data can be described by equations. Many discovery systems assume functional relationships in data, and focus on identifying the best functional expression. This assumption is justified if the data come from well designed, high quality experiments in physics or chemistry, but is seldom satisfied in most other cases. However, when strong functional relationships are possible, it would be a big fault not to notice them when they occur. We need a test which applies to all datasets and is passed only when the data can be approximated by a function.

We will present such a test, which determines whether there is a functional dependency between two given variables: independent (called x) and dependent (y). If the dependent variable cannot be identified, the test should be run twice, first for y as dependent and then for x as dependent.

2.1 The Definition of Functionality

We will use the following mathematical definition of functional relationship:

Definition: Given a set DB of pairs (x_i, y_i), $i = 1, \ldots, N$ of two variables x and y, and the range X of x, y is a function of x iff for each x_0 in X, there is exactly one value of y, say y_0, such that (x_0, y_0) is in DB.

We could add a second requirement: y should be a continuous function of x. However, discrete data make testing continuity difficult and missing data frequently results in artificial discontinuities. After conducting many experiments we decided not to use analysis of continuity in our functionality test. The issue of continuity is re-visited in the section on multiple equation finder.

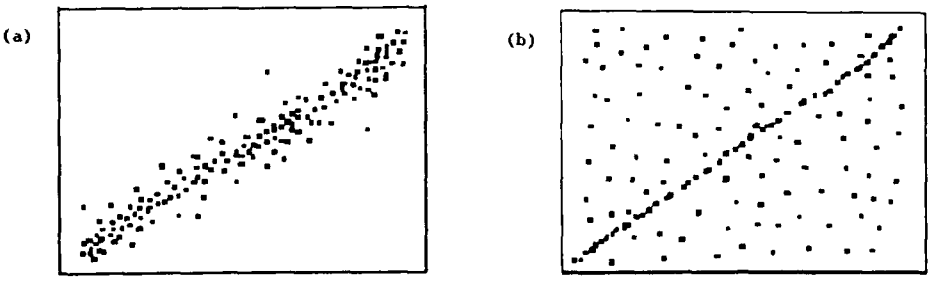
2.2 Problems with Functionality in Real Data

It would be easy to check functionality in a given dataset by the strict application of the definition, that is, by checking whether indeed a unique value of x corresponds to one value of y. However, in real world data, because of phenomena collectively called error and noise, strict uniqueness does not occur and

therefore the condition is unrealistic. We want a more relaxed test that will admit data sets which can be reasonably approximated by equations. In particular, we want to admit a deviation in values, so rather than require a single value of y permit a distribution of values in a relatively small interval. This fits a statistical model $y = f(x) + \delta(x)$, where $f(x)$ describes relationship between x and y, $\delta(x)$ incorporates effects of phenomenas like error or noise.

Figure 1a illustrates how error complicates the problem: for the same value of x there could be many different values of y. A test strictly based on the definition would frequently fail *even* if the data could be described by a function within a small deviation of Y. If the error δ_y of y is known, the comparison of values of y should be related to the error. For example, one can compare $|y_1 - y_2|$ versus $\delta_{y_1} + \delta_{y_2}$ (δ_{y_i} is the error of y_i). It means that the definition of the functionality should be modified for our purpose: for each value of x, differences between the corresponding values of y must be in the range $(-\delta_y, \delta_y)$. However, in many cases error is unknown. Error is typically estimated only for experimental data.

Fig. 1. (a) Effect of error: there is many points that have the same value of x but different y values. Because these data can't be described by a (linear) function plus a small error, we need a more permissive definition of functionality. (b) Effect of noise: in addition to data that follow a functional dependency, additional distribution of points is not correleated with that dependency. These additional points clearly violate the definition of functionality, even if tolerance to small error is introduced, because in many cases the differences between the values of y for one value of x are large). However, a linear dependency in data still can be easily noticed.



Another important factor is background noise, which comes from many sources and is common in databases. Even if all combinations of values are present in the data due to the noise, the functional relationship may still stand out by higher concentration of data. Figure 1b shows background noise added to a functional dependency: in addition to data that can be well described by a linear function, a large number of points is randomly distributed.

Possibility of missing data is another factor that adds to the complexity of the problem. For some values of x, data on the functional dependence may not be

present. This is common in databases because in many cases there is no control over data collection process.

Finally, we want to tolerate a small number of outliers, that is data which stand out from noise and lie drastically outside the error distribution. Such data could be entered by error, or they represent a phenomenon of potential interest. Still we wish to neglect those data if they are too few to make sense of them.

2.3 Determination of the Uniqueness

In databases, typically a small discrete set of values is permitted for each attribute. Given a small set X of x values, and a small set Y of y values, the product $X \times Y$ is computationally manageable, as well as the corresponding frequency table F which is a mapping $F : X \times Y \rightarrow N$, where N is the set of natural numbers, and $F(x_0, y_0) = n$ when n is the number of datapoints with $x = x_0$ and $y = y_0$. If the number of values of x and/or y becomes too large to compute the frequency table, the values of x and y can be grouped into bins $b(X)$ and $b(Y)$, respectively.

Aggregating the values of x into bins $b(X)$ of equal size Δx means that the point (x_0, y_0) is replaced by a pair of integer numbers (k_x, k_y) such that x_0 is in the range from $x_{min} + k_x \Delta x$ to $x_{min} + (k_x + 1)\Delta x$, and y_0 is in the range from $y_{min} + k_y \Delta y$ to $y_{min} + (k_y + 1)\Delta y$, where x_{min} and y_{min} are the smallest values of x and y, respectively. Note that the frequency table F can be interpreted as a grid $b(X) \times b(Y)$ imposed on the data (x, y) and defined by Δx and Δy.

Binning the original variables x and y into bins $b(X)$ and $b(Y)$ is very helpful for determining functionality in data which include error and/or noise. If the grid size Δy is comparable to the error δ_y, the requirement for diffences between y values corresponding to the same value of x can be replaced by the following: all points lying in the same x-bin k_x must lie in the adjacent y-bins (for example, in bins $k_y - 1$, k_y, $k_y + 1$). Note that the problem of error is removed only if the bin sizes Δx and Δy are not smaller than corresponding errors; otherwise it may still happen that the functionality test fails because points in the same x-bin may not lie in *adjecent* y-bins, even if the original data really follow a functional dependency plus error. On the other hand, if the sizes Δx and Δy are too large, the test could assign functionality to data that intuitively should not be described by a single function. In the extreme case, when Δy is larger than $y_{max} - y_{min}$, all points always lie in the same y-bin, because one includes all values of y.

The problem of background noise can be at least alleviated if instead of checking adjacency of non-empty cells $F(k_x, k_y)$ which have the same value k_x, one considers only cells that contain an above average number of points. This noise-subtraction works effectively only if the background noise is not stronger than the regularity itself.

If the errors δ_x and δ_y are known, they can be used as the corresponding bin sizes Δx and Δy. But if they are unknown or uncertain, the proper grid sizes must be estimated.

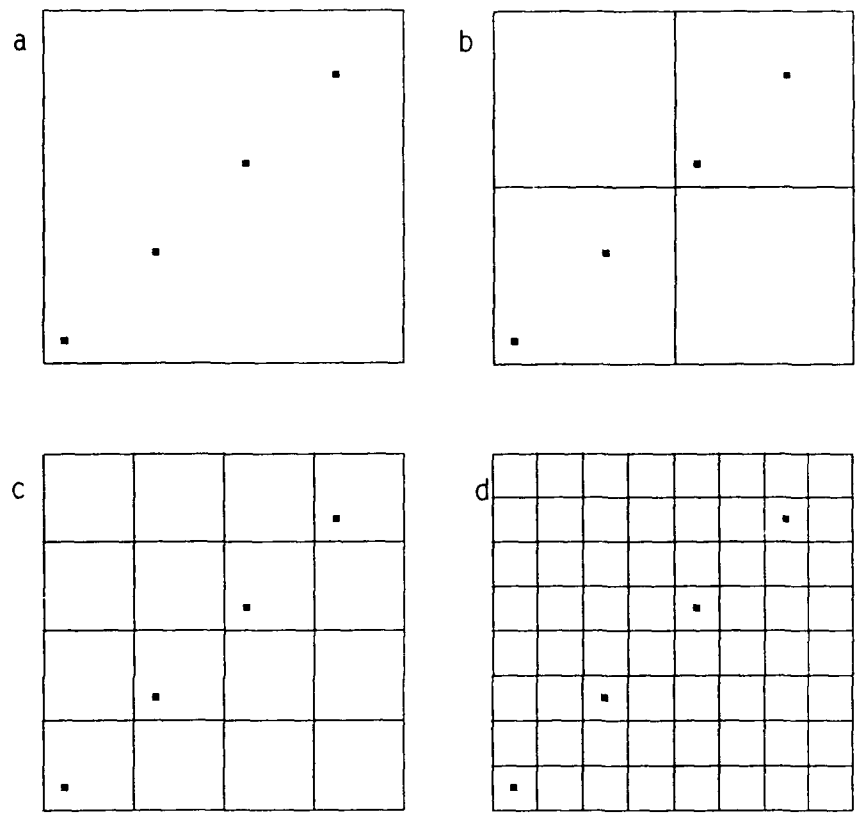
Before we present the algorithm which estimates the grid sizes, let us consider an ideal functional dependence, say a straight line, as in Figure 2a: the data are evenly distributed with constant distance Δ in x between points. Let us define the "density" ρ as the average number of points in all cells which contain at least one point. As long as the grid size Δx is smaller than Δ, ρ is equal to one. The density ρ starts to grow when Δx becomes greater than Δ. Note that as opposed to the true density equal to the number of data points divided by the total number of cells, ρ does not depend on the number of x-bins in the following sense. If we extend the range of x by adding more points which are evenly distributed, and keeping Δx constant, then in our ideal example, ρ will have exactly the same values. This is important because it means that the "density" measure ρ does not depend on the range of x or y as long as bin sizes are the same. Therefore ρ can be used to determine Δx and Δy.

Determination of the Grid Size. Our algorithm for finding the grid size starts from some small initial sizes Δx and Δy and then changes these sizes until a criterion for density of points is satisfied.

Algorithm: Determine grid size
  $\Delta x \leftarrow X/2N\rho_0$, $\Delta y \leftarrow Y/2N\rho_0$
  $\rho \leftarrow N$ / (# of non-empty cells)
  if $\rho \leq \rho_0$ **then**
    repeat
      $\Delta x \leftarrow 2\Delta x$, $\Delta y \leftarrow 2\Delta y$
      $\rho \leftarrow N$ / (# of non-empty cells)
    until $\rho > \rho_0$
  else
    repeat
      $\Delta x \leftarrow \Delta x/2$, $\Delta y \leftarrow \Delta y/2$
      $\rho \leftarrow N$ / (# of non-empty cells)
    until $\rho < \rho_0$
    $\Delta x \leftarrow 2\Delta x$, $\Delta y \leftarrow 2\Delta y$
  end if
end algorithm

ρ_0 is the minimum required "density" of points in non-empty cells. The initial values of Δx and Δy are chosen to be $X/2N\rho_0$ and $Y/2N\rho_0$, respectively, because for monotonic functions with more or less evenly distributed points the resulting density ρ would be close to ρ_0. The additional factor $1/2$ was introduced to avoid decreasing the grid size in cases when initial ρ is only slightly larger than ρ_0; note that decreasing the grid size is more costly than increasing it — because the latter operation can be performed on the existing grid from the previous step, while in the former case the density grid must be build from data. From the definition of ρ one can see that its value is never smaller than 1. If the grid size is too small, the value of ρ is about 1. When ρ starts to grow and becomes significantly greater than 1, say its value is about 2, it means that there is on average about two points per each non-empty cells. At that moment for most

Fig. 2. Comparison of density for cells which contain data (ρ) and true density: (a) sample points lying along a straight line $y = ax + b$; (b) grid with $\Delta x = \Delta/2$, $\rho = 1$, density $= 1/16$; (c) grid with $\Delta x = \Delta$, still $\rho = 1$ but density $= 1/4$; (c) grid with $\Delta x = 2\Delta$, $\rho = 2$ and density $= 1$. Note that parameter ρ stays the same until Δx becomes larger than Δ or Δy larger than $b\Delta$; Δ is the smallest difference between x coordinates of points.



data one can start analyzing functionality based on that grid size, therefore the best default value for ρ_0 is around 2. This default value works very well for more or less evenly distributed data points. However, when the distribution of data is very uneven, ρ_0 should be increased.

2.4 Functionality Test

The input to the functionality test is a frequency table of data for a grid based on $b(X)$ and $b(Y)$, the binned values of x and y. The grid $b(X) \times b(Y)$ can be (1) built based on the known error, (2) determined by the above algorithm, or (3) obtained from the already binned data. The following algorithm determines whether it is worthwhile to search for an equation that fits the data:

Algorithm: Test functional relationship between x and y
given the table of actual record counts
AV ← average number of records per cell
for each value in b(X)
find all groups of adjacent cells with counts > AV
if # of groups > α **then**
return NO-FUNCTION
end if
end for
if average # of groups > β **then**
return NO-FUNCTION
else
return FUNCTION
end if
end algorithm

The algorithm is controlled by two modifiable parameters, α and β, which are measures of local (α) and global (β) uniqueness in y, that correspond to the number of values of y for the same value of x. The default values used by 49er are $\alpha \approx 3$, $\beta \approx 1.5$. For $\alpha = 3$ the *functionality test fails when for a value* in $b(X)$ there is more than 3 adjacent groups of cells with the above average density of points. This higher value of α solves the problem of rare outliers: there could be up to 2 outliers (that is, cells having an above average number of points) provided it happens very rarely. However, many outliers or frequent discontinuities in y should fail the test, therefore the value of β should be much smaller and close to one.

The values of α and β should be increased for sparse data with strong background noise, while they should be reduced if we are looking for high-precision functional dependencies in experimental data. If the background noise is strong, fluctuations in noise could results in cells having above the average number of points coming from pure noise. In that case, considering cells with the above average density of points may not always be a cure for the noise. In such cases one should increase the values of α and β to allow more discontinuities in y, coming from noise. On the other extreme, if the data result from high-precision experiments with very little noise, discontinuities in y usually would reflect the true nature of the dependency in data (if any) and therefore should be treated more seriously: α and β should be decreased.

3 An Application: Discovery of Functionality in 49er

49er is a machine discovery system that analyzes large databases in search for regularities hidden in those data. In the first phase of its exploration of a database, 49er looks for regularities holding for 2 attributes (variables). We will focus on that problem.

The two basic types of regularities considered by 49er are contingency tables and equations. While analysis of contingency tables is relatively fast (for exam-

ple, 49er needs about 0.1s for a typical contingency-all regularity), the search for equations is quite costly (3-5 seconds for the same data). Therefore 49er uses domain knowledge about attributes (for example, to avoid search for equations for nominal attributes) and the test for functionality to avoid the search for equations when there is no hope to find them.

Table 1 presents empirical data which show advantages of the functionality test on several test runs. While the number of detected regularities was only slightly smaller when the test has been applied, the computation time was significantly shorter. In tests 1–4 some 26,000 of datasets were examined by 49er. Only few of them can be reasonably approximated by functional dependencies. Most discovered equations are very weak. Equations "lost" when the test was used belong to the weakest of all discovered equations and their number varies when one slightly changes α and β. Test 5 and 6 demonstrate the very small overhead introduced by the application of the test: a quite strong functional dependence was in all datasets. Note that this time no equation was lost. Many tests conducted on real and artificially generated data show that only a small percent of very weak equations could be sometimes lost when the test is used.

Table 1. Comparison of 49er execution times with and without test for functionality. Tests 1–4 were conducted on about *1400 records*, *10 attributes*, and an average 10 values per attribute. 49er considered about 26,000 datasets in those data. The results show that run-time was reduced significantly while only about 10% of equations were not discovered. Tests 5 and 6 were applied to about 6000 records, 6 attributes and from 3 to 50 values per attribute. In tests 5 and 6 49er focused only on pairs of attributes that had some functional dependencies, known or discovered earlier, so that the test almost always resulted in the "yes" answer. Note that execution times for tests 5 and 6 are almost the same — it means that the overhaed introduced by the test for functionality can be neglected compared to time needed by Equation Finder.

Test number	Functionality test used?	Number of equations	CPU time	Comments
1	yes	71	258 minut	database of
2	no	77	610 minut	1400 records
3	yes	24	58 minut	same data but a more
4	no	26	80 minut	shallow search
5	yes	23	699 second	6000 records, only datasets
6	no	23	695 second	with functional dependencies

The functionality test can be also applied as a tool for discovering functional dependencies between attributes. When requested, 49er can use only this test while analyzing a database. The results are of interest to users looking only for the existence of functional dependencies in data, but can also be used to modify the default search strategy of 49er. The latter application is especially useful for databases with many attributes: the functionality test can be quickly applied to many pairs of attributes. The results of this preliminary search can be used to direct the search in the most promising directions, without considering combinations of attributes for which the functionality test failed or returned high values of β.

4 Discovery of Multifunctions

We will now discuss an extension of the functionality test that leads to discovery of multifunctions. Given data pass our functionality test when for most values of x there is only one continuous group of cells with counts above the average. However, if the test consistently shows two groups of cells for a subrange of x, it could mean that there are *two* functional dependencies in data: $y_1 = y_1(x)$ and $y_2 = y_2(x)$.

The existence of two or more functional dependencies in data is practically useful. For example, if one collects data being weight of African elephants of different age, one can observe that there seems to be two distinct dependencies weight–age. A closer look could show that there are actually two kinds of African elephants: large bush elephants (*Loxodonta africana*) and smaller forest elephants (*Loxodonta cyclotis*). Such a discovery of two or more functional dependencies in data could lead to the discovery of a classification cite Piatetsky-Shapiro.

We have implemented Multiple Equation Finder (called MEF) that uses an extended version of the functionality test to discover many functional dependencies. If the test returns high value β, MEF checks continuity of cell groups in x direction, partitions data, runs Equation Finder for every group of adjacent cells, and finally tries to extend range of discovered equations (if any) and reduce number of groups by merging them if they can be described simultaneously by means of the same equation. Detailed description of group merging goes beyond the scope of this paper and will be presented in another article.

Algorithm: Multiple Equation Finder

```
run Test for Functionality
if average # of groups < β then
   run Equation Finder
else
   for each group of adjacent high-density cells
      merge the group with others if they are adjacent in x
   end for
   for every group spanning through many subranges of x
      run Equation Finder
   end for
   repeat
      merge two groups when they have similar equations or
            equation of one groups describes data in the other group
   until no merging happened in the last iteration
end if
end algorithm
```

5 Conclusions

In this paper we have presented a test which detects existence of functional dependencies in data. Currently, the functionality test is used (1) in the database

mining system 49er as an application condition of the search for equations and (2) to find multiple equations in data coming from databases. In 49er, the test for functionality is always applied to data before Equation Finder search starts searching for equations. In Multiple Equation Finder, the test determines whether and how the data should be partitioned so that a single equation can become a search target in each partition.

References

1. B.C. Falkenhainer & R.S. Michalski: Integrating quantitative and qualitative discovery: the ABACUS system. *Machine Learning* 1, pp367–422 (1986)
2. P. Hoschka & W. Klösgen: A Support System for Interpreting Statistical Data, in: Piatetsky-Shapiro G. & Frawley W. eds *Knowledge Discovery in Databases*, Menlo Park, Calif.: AAAI Press (1991)
3. W. Klösgen: Patterns for Knowledge Discovery in Databases in: Żytkow J. ed *Proceedings of the ML-92 Workshop on Machine Discovery (MD-92)*, National Institute for Aviation Research, Wichita, Kansas, pp.1–10 (1992)
4. P. Langley, H.A. Simon, G.L. Bradshaw, & J.M. Żytkow: *Scientific discovery: Computational explorations of the creative processes.* Cambridge, MA: MIT Press (1987)
5. M. Moulet: ARC.2: Linear Regression In ABACUS, in: Żytkow J. ed *Proceedings of the ML-92 Workshop on Machine Discovery (MD-92)*, National Institute for Aviation Research, Wichita, Kansas, pp.137–146 (1992)
6. B. Nordhausen & P. Langley: An Integrated Approach to Empirical Discovery. in: J.Shrager & P. Langley (eds.) *Computational Models of Scientific Discovery and Theory Formation*, pp. 97–128, Morgan Kaufmann Publishers, San Mateo, CA (1990)
7. G. Piatetsky-Shapiro & C. Matheus: Knowledge Discovery Workbench, in: G. Piatetsky-Shapiro ed. *Proc. of AAAI-91 Workshop on Knowledge Discovery in Databases*, pp. 11–24 (1991)
8. R. Zembowicz & J.M. Żytkow: Automated Discovery of Empirical Equations from Data, *Proceedings of the ISMIS-91 Symposium*, Springer-Verlag (1991)
9. R. Zembowicz & J.M. Żytkow: Discovery of Regularities in Databases, in Żytkow J. ed *Proc. ML-92 Workshop on Machine Discovery.* Aberdeen, U.K. pp. 18–27 (1992)
10. J. Żytkow & J. Baker: Interactive Mining of Regularities in Databases. In *Knowledge Discovery in Databases*, eds. G. Piatetsky-Shapiro and W. Frawley. Menlo Park, Calif.: AAAI Press (1991)
11. J.M. Żytkow: Combining many searches in the FAHRENHEIT discovery system, *Proceedings of the Fourth International Workshop on Machine Learning.* Irvine, CA: Morgan Kaufmann, 281-287 (1987).

ROUGH SET LEARNING OF PREFERENTIAL ATTITUDE IN MULTI-CRITERIA DECISION MAKING

Roman Słowiński

Institute of Computing Science, Technical University of Poznań
60-965 Poznań, Poland

Abstract. Rough set theory is a useful tool for analysis of decision situations, in particular multi-criteria sorting problems. It deals with vagueness in the representation of decision maker's (DM's) preferences, caused by granularity of the representation. Using the rough set approach, it is possible to learn a set of sorting rules from examples of sorting decisions made by the DM. The rules involve a minimum number of most important criteria and they do not correct vagueness manifested in the preferential attitude of the DM; instead, produced rules are categorized into deterministic and non-deterministic. The set of sorting rules explains a decision policy of the DM and may be used to support next sorting decisions. The decision support is made by matching a new case to one of sorting rules; if it fails, a set of the 'nearest' sorting rules is presented to the DM. In order to find the 'nearest' rules a new distance measure based on a valued closeness relation is proposed.

1 Introduction

Decision making is one of the most natural acts of human beings. Decision analysis has attracted scientists for a long time who offered various mathematical tools to deal with. Scientific decision analysis intends to bring into light those elements of a decision situation which are not evident for the actors and may influence their attitude towards the situation. More precisely, the elements revealed by the scientific decision analysis either explain the situation or prescribe, or simply privilege, some behavior in order to increase the coherence between evolution of the decision process on the one hand and goals and value systems of the actors, on the other hand (cf. Roy (1985)).

When making real decisions, the decision maker (DM) usually takes into account multiple points of view (criteria) for evaluation of decision alternatives. However, the multi-criteria decision problem has no solution without additional information about DM's preferences. Having the preferential information, one can build on the multiple criteria a global preference model which yields 'the best' solution of a given decision problem.

There are two major ways of constructing a global preference model upon

preferential information obtained from a DM. The first one comes from mathematical decision analysis and consists in building a functional or a relational model (cf. Roubens and Vincke (1985)). The second one comes from artificial intelligence and builds up the model via learning from examples (cf. Michalski (1983)). In the context of artificial intelligence, the preferential information is called *knowledge* about preferences and the DM is often called an *expert*.

In this paper, we are interested in the second way of constructing a global preference model for a *multi-criteria sorting problem*. The sorting problem consists in assignment of each object from a set to an appropriate pre-defined category (for instance: acceptance, rejection or request for an additional information). In this case, the global preference model consists of a set of logical statements (*sorting rules*) "if ... then ..." describing a preferential attitude of the DM. The set of rules is derived from examples of sorting decisions taken by the DM (expert) on a subset of objects. The multi-criteria sorting of a new object is then supported by matching its description to one of the sorting rules; if it fails, a set of the 'nearest' sorting rules (in the sense of a given distance measure) is presented to the DM.

The preferential information is usually vague (inconsistent) because of different sources of uncertainty and imprecision (cf. Roy (1989)). Vagueness may be caused by granularity of a representation of preferential information. Due to the granularity, the rules describing a preferential attitude of the DM can be categorized into *deterministic* and *non-deterministic*. Rules are deterministic if they can be described univocally by means of 'granules' of the representation, and they are non-deterministic, otherwise.

A formal framework for discovering deterministic and non-deterministic rules from a given representation of knowledge has been given by Pawlak (1982) and called *rough set theory*. The rough set theory assumes knowledge representation in a *decision table* form which is a special case of an information system. Rows of this table correspond to *objects* (actions, alternatives, candidates, patients, etc.) and columns correspond to *attributes*. For each pair (object, attribute) there is known a value called *descriptor*. Each row of the table contains descriptors representing information about corresponding object from the universe. In general, the set of attributes is partitioned into two subsets: *condition attributes** (criteria, features, symptoms, etc.) and *decision attributes* (decisions, assignments, classifications, etc.).

The observation that objects may be indiscernible in terms of descriptors is a starting point of the rough set philosophy. Indiscernibility of objects by means of attributes prevents generally their precise assignment to a set. Given an equivalence relation viewed as an indiscernibility relation which thus induces an approximation space made of equivalence classes, a *rough set* is a pair of a

*In decision problems, the concept of criterion is often used instead of condition attribute. It should be noticed, however, that the two concepts have sometimes different meaning because the domain (scale) of a criterion has to be ordered according to decreasing or increasing preference while the domain of a condition attribute has not to be ordered. Similary, the domain of a decision attribute may be ordered or not.

lower and an upper approximation of a set in terms of the classes of indiscernible objects. In other words, a rough set is a collection of objects which, in general, cannot be precisely characterized in terms of the values of the set of attributes, while a lower and an upper approximation of the collection can be. Using a lower and an upper approximation of a set (or family of sets - partition) one can define an accuracy and a quality of approximation. These are numbers from interval $[0, 1]$ which define how exactly one can describe the examined set of objects using available information. The most complete presentation of the rough set theory can be found in Pawlak (1991).

We shall use the rough set approach to derive sorting rules from examples given by the DM. In the next section, we characterize the methodology, including the use of sorting rules for decision support. In section 3, we apply the proposed methodology to an example of selection of candidates to a school. The final section groups conclusions.

2 Rough Set Approach to a Multi-Criteria Sorting Problem

2.1 Problem definition and expected results

Having a set of objects described by a number of criteria, the sorting problem consists in assignment of each object to an appropriate pre-defined category (for instance: acceptance, rejection or request for an additional information).

Let the preferential information coming from a DM, or an expert, be given in the form of a decision table where objects correspond to examples, condition attributes to criteria and the only decision attribute, to decisions about assignment to a category. In other words, the rows of the table are examples of sorting decisions related to the corresponding objects.

We are expecting the following results from the rough set analysis of the above preferential information:

- evaluation of importance of particular criteria,
- construction of minimal subsets of independent criteria ensuring the same quality of sorting as the whole set, i.e. reducts of the set of criteria,
- non-empty intersection of those reducts gives a core of criteria which cannot be eliminated without disturbing the ability of approximating the sorting decisions,
- elimination of redundant criteria from the decision table,
- generation of the sorting rules from the reduced decision table; they involve the relevant criteria only and explain a decision policy of the DM (expert).

Of course, the most important result from the viewpoint of decision support is the set of sorting rules. It constitutes a global model of DM's (expert's) preferences based on the set of examples. Using the terms from AI, the set of

examples is a training sample for learning of the expert's preferential attitude (cf. Grzymala-Busse (1992)).

Rough set approach has been applied with success to sorting problems from medicine (Fibak et al. (1986), Slowinski et al. (1988)), pharmacy (Krysinski (1990)), technical diagnostics (Nowicki et al. (1992)) and many others (the most complete reference can be found in Slowinski, ed. (1992)).

2.2 Decision support using sorting rules

The sorting of a new object can be supported by matching its description to one of the sorting rules. The matching may lead to one of four situations:

(i) the new object matches exactly one of deterministic sorting rules,

(ii) the new object matches exactly one of non-deterministic sorting rules,

(iii) the new object doesn't match any of the sorting rules,

(iv) the new object matches more than one rule.

In (i), the sorting suggestion is direct. In (ii), however, the suggestion is no more direct since the matched rule is ambiguous. In this case, the DM is informed about the number of sorting examples which support each possible category. The number is called a *strength*. If the strength of one category is greater than the strength of other categories occurring in the non-deterministic rule, one can conclude that according to this rule, the considered object most likely belongs to the strongest category.

Situation (iii) is more burdensome. In this case, one can help the DM by presenting him a set of the rules 'nearest' to the description of the new object. The notion of 'nearest' involves the use of a distance measure. In this paper, we present one known distance measure and define a new one having some good properties.

As a sorting rule may have less conditions (criteria) than the description of an object to be sorted, the distance between the sorting rule and the object will be computed for criteria represented in the rule only. So, it is assumed that there is no difference for other criteria.

Let a given new object x be described by values $c_1(x), c_2(x), \ldots, c_m(x)$ ($m \leq card(C)$) occurring in the condition part of the sorting rule y described by values $c_1(y), c_2(y), \ldots, c_m(y)$ of the same m criteria.

According to the first definition (cf. Stefanowski (1992), Slowinski and Stefanowski (1992)), the distance of object x from rule y is equal to

$$D = 1/m\Big\{\sum_{l=1}^{m}[k_l[\mid c_l(x) - c_l(y) \mid /(v_{l\max} - v_{l\min})]^p\Big\}^{1/p}$$

where: $p = 1, 2, \ldots$ - natural number to be chosen by an analyst,
$v_{l\max}, v_{l\min}$ - maximal and minimal value of c_l , respectively,
k_l - importance coefficient of criterion c_l .

Let us notice that depending on the value of p, the distance measure is more or less compensatory. Moreover, the greater is the value of p the greater is the importance of the largest partial difference in D. The importance coefficients can be determined either subjectively or taking into account a sorting ability of criteria, e.g. the difference between the quality of sorting for the set of all considered criteria and the quality of sorting for the same set of criteria not including the checked one.

Some restrictions on the distance between a given object and the rule can also be introduced, e.g. a maximal number of differences on corresponding positions of criteria.

The above definition has, however, some weak points. In particular, for low values of p, it allows a major difference on one criterion to be compensated by a number of minor differences on other criteria. For high values of p, it has a tendency to overvalue the greatest partial difference, up to the point of ignoring all other partial differences, for $p = \infty$.

This criticism leads us to definition of a new distance measure based on, so called, *valued closeness relation* R. It is inspired by the outranking relation introduced by Roy (*1985*).

The new object x will be compared to each sorting rule y in order to assess the credibility of the affirmation: "x is close to y", what is denoted by xRy. The calculation of the credibility $r(x, y)$ of this affirmation is based on common-sense: the formula determining the value of $r(x, y)$ over the interval $[0, 1]$ is constructed so as to respect certain qualitative principles, and, in particular, excludes the possibility of undesired compensation. Credibility $r(x, y) = 1$ if the assertion xRy is well-founded; $r(x, y) = 0$ if there is no argument for closeness of x to y. The formula for calculation of $r(x, y)$ is essentially based on two concepts called *concordance* and *discordance*. The goals of these concepts are to:

- characterize a group of criteria (conditions) considered to be in concordance with the affirmation being studied and assess the relative importance of this group of criteria compared with the remainder of the m criteria,
- characterize among the criteria which are not in concordance with the affirmation being studied, the ones whose opposition is strong enough to reduce the credibility which would result from taking into account just the concordance, and to calculate the possible reduction in it that would thereby result.

To be able to carry out such calculations, one must first express explicitly and numerically:

- the relative importance k_l that the DM wishes to confer on criterion c_l in the calculation of concordance,
- the minimum value of discordance which gives criterion c_l the power to take all credibility away from the affirmation of the closeness, even if opposed to all the other criteria in concordance with the affirmation; it is denoted by $v_l[c_l(x)]$ and called the veto *threshold* of criterion c_l.

The global concordance index is defined as

$$C(x,y) = \sum_{l=1}^{m} k_l c_l(x,y) / \sum_{l=1}^{m} k_l$$

where $c_l(x,y)$ is a partial concordance index for criterion c_l . Calculation of $c_l(x,y)$ involves two thresholds: $0 \leq q_l[c_l(x)] \leq p_l[c_l(x)]$, called *indifference* and *strict difference thresholds*, respectively. The definition of $c_l(x,y)$ and discordance index $D_l(x,y)$ for criterion c_l is given graphically in Fig.1.

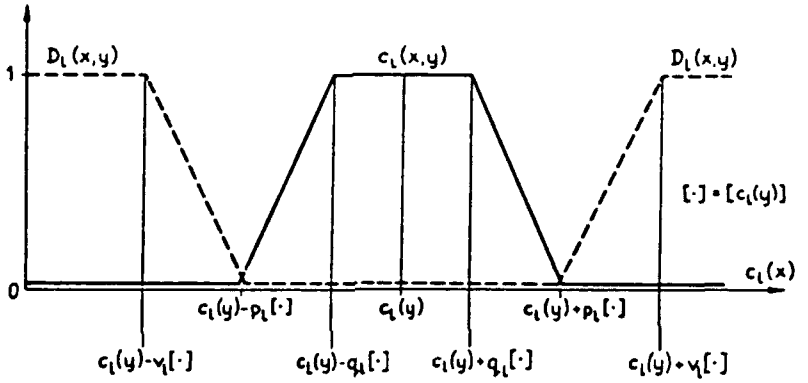


Fig.1. Concordance and discordance indices for object x and rule y, with respect to criterion c_l

The degree of credibility $r(x,y)$ of the closeness relation xRy is obtained from the global concordance index weakened by discordance indices (up to the point of its annulment):

$$r(x,y) = C(x,y) \prod_{l \in L} \frac{1 - D_l(x,y)}{1 - C(x,y)}, \quad L = \{l : D_l(x,y) > C(x,y)\}$$

The rules y with the greatest values of $r(x,y)$ are presented to the DM together with an information about the strength of the corresponding categories.

Situation (iv) may also be ambiguous if the matched rules (deterministic or not) lead to different categories. Then, the suggestion can be based either on the strength of possible categories, or on an analysis of the sorting examples which support each possible category. In the latter case, the suggested category is that one which is supported by a sorting example being the closest to the new object, in the sense of relation R.

3 Selection of Candidates to a School

To illustrate the rough set learning of a preferential attitude, let us consider a simple case of selection of candidates to a school (cf. Moscarola (1978)).

The candidates to the school have submitted their application packages with secondary school certificate, curriculum vitae and opinion from previous school, for consideration by an admission committee. Basing on these documents, the candidates were described using seven criteria (condition attributes). The list of these criteria together with corresponding scales, ordered from the best to the worst value, is given below:

c_1 - score in mathematics, {5,4,3}
c_2 - score in physics, {5,4,3}
c_3 - score in English, {5,4,3}
c_4 - mean score in other subjects, {5,4,3}
c_5 - type of secondary school, {1,2,3}
c_6 - motivation, {1,2,3}
c_7 - opinion from previous school, {1,2,3}

Fifteen candidates having rather different application packages have been sorted by the committee after due consideration. This set of examples will be used to learn a preferential attitude of the committee.

The decision attribute d makes a dichotomic partition $\mathcal{Y}$ of the candidates: $d = A$ means admission and $d = R$ means rejection. The decision table with fifteen candidates is shown in Table 1. It is clear that $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$ and $D = \{d\}$.

Let Y_A be the set of candidates admitted and Y_R the set of candidates rejected by the committee, $Y_A = \{x_1, x_4, x_5, x_7, x_8, x_{10}, x_{11}, x_{12}, x_{15}\}$, $Y_R = \{x_2, x_3, x_6, x_9, x_{13}, x_{14}\}$, $\mathcal{Y} = \{Y_A, Y_R\}$. Sets Y_A and Y_R are D-definable sets in the decision table. There are 13 C-elementary sets: couples of indiscernible candidates $\{x_4, x_{10}\}$, $\{x_8, x_9\}$ and 11 discernible candidates. The C-lower and the C-upper approximations of sets Y_A and Y_R are equal, respectively, to:

$$\underline{C}Y_A = \{x_1, x_4, x_5, x_7, x_{10}, x_{11}, x_{12}, x_{15}\}$$

$$\overline{C}Y_A = \{x_1, x_4, x_5, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{15}\}$$

$$Bn_C(Y_A) = \{x_8, x_9\}$$

$$\underline{C}Y_R = \{x_2, x_3, x_6, x_{13}, x_{14}\}$$

$$\overline{C}Y_R = \{x_2, x_3, x_6, x_8, x_9, x_{13}, x_{14}\}$$

$$Bn_C(Y_R) = \{x_8, x_9\}$$

The accuracy of approximation of sets Y_A and Y_R by C is equal to 0.8 and 0.71, respectively, and the quality of approximation of the decision by C is equal to 0.87.

Let us observe that the C-doubtful region of the decision is composed of two candidates: x_8 and x_9. Indeed, they have the same value according to criteria from C but the committee has admitted x_8 and rejected x_9. It means that the decision is inconsistent with evaluation of the candidates by criteria from C. So, apparently, the committee took into account an additional information from the application packages of the candidates or from an interview with them. This

Table 1. Decision table composed of sorting examples

Criterion / Candidate	c_1	c_2	c_3	c_4	c_5	c_6	c_7	Decision d
x_1	4	4	4	4	2	2	1	A
x_2	3	3	4	3	2	1	1	R
x_3	3	4	3	3	1	2	2	R
x_4	5	3	5	4	2	1	2	A
x_5	4	4	5	4	2	2	1	A
x_6	3	4	3	3	2	1	3	R
x_7	4	4	5	4	2	2	2	A
x_8	4	4	4	4	2	2	2	A
x_9	4	4	4	4	2	2	2	R
x_{10}	5	3	5	4	2	1	2	A
x_{11}	5	4	4	4	1	1	2	A
x_{12}	5	3	4	4	2	2	2	A
x_{13}	4	3	3	3	3	2	2	R
x_{14}	3	3	4	3	2	3	3	R
x_{15}	4	5	5	4	2	1	1	A

conclusion suggests to the committee, either adoption of an additional discriminatory criterion or, if its explicit definition would be too difficult, creation of a third category of candidates : those who should be invited to an interview.

The next step of the rough set analysis of the decision table is construction of minimal subsets of independent criteria ensuring the same quality of sorting as the whole set C, i.e. the reducts of C. In our case, there are three such reducts:

$$RED^1_{\mathcal{Y}}(C) = \{c_2, c_3, c_6, c_7\}$$

$$RED^2_{\mathcal{Y}}(C) = \{c_1, c_3, c_7\}$$

$$RED^3_{\mathcal{Y}}(C) = \{c_2, c_3, c_5, c_7\}$$

It can be said that the committee took the fifteen sorting decisions taking into account the criteria from one of the reducts and discarded all the remaining criteria. Let us notice that criterion c_4 has no influence at all on the decision because it is not represented in any reduct.

It is interesting to see the intersection of all reducts, i.e. the core of criteria:

$$CORE_{\mathcal{Y}}(C) = RED^1_{\mathcal{Y}}(C) \cap RED^2_{\mathcal{Y}}(C) \cap RED^3_{\mathcal{Y}}(C) = \{c_3, c_7\}$$

The core is the most essential part of set C, i.e. it cannot be eliminated without disturbing the ability of approximating the decision.

In a real case, all the reducts and the core should be submitted for consideration by the committee in view of getting its opinion about what reduct should be used to generate sorting rules from the reduced decision table.

Let us suppose that the committee has chosen reduct $RED^2_{\mathcal{Y}}(C)$ composed of c_1, c_3, c_7 , i.e. scores in mathematics and English, and opinion from previous school. This choice could be explained in such a way that the score in mathematics (c_1) seems to the committee more important than the score in physics (c_2) plus type of secondary school (c_5) or motivation (c_6).

Now, the decision table can be reduced to criteria represented in $RED^2_{\mathcal{Y}}(C)$. The sorting rules generated from the reduced decision table have the following form:

rule #1:	if $c_1 = 5$						then d=A
rule #2:	if		$c_3 = 5$				then d=A
rule #3:	if $c_1 = 4$			and	$c_7 = 1$		then d=A
rule #4:	if $c_1 = 4$	and	$c_3 = 4$	and	$c_7 = 2$		then d=A or R
rule #5:	if $c_1 = 3$						then d=R
rule #6:	if		$c_3 = 3$				then d=R

Five rules are deterministic and one is non-deterministic. The non-deterministic rule #4 follows from indiscernibility of candidates x_8 and x_9 which belong to different categories of decision. It defines a profile of candidates which should create the third category of decision, e.g. those candidates who should be invited to an interview.

The rules represent clearly the following policy of the selection committee:

> *Admit all candidates having score 5 in mathematics or in English. Admit also those who have score 4 in mathematics and in English but very good opinion from a previous school. In the case of score 4 in mathematics and in English but only a moderate opinion from a previous school, invite the candidate to an interview. Candidates having score 3 in mathematics or in English are to be rejected.*

The sample of fifteen sorting decisions have been considered as a training sample and used to reveal the preferential attitude of the committee. The global preference model represented by the set of 6 sorting rules could be applied now to support selection of new candidates, as suggested in p.2.2. of this paper.

4 Concluding Remarks

The aim of this paper was to show that the rough set theory is a useful tool for discovery of a preferential attitude of the DM in multi-criteria decision making problems, in particular multi-criteria sorting problems.

We claim that the global preference model in the form of rules derived from a set of examples has an advantage over a functional or a relational model because it explains the preferential attitude through important facts in terms of significant criteria only. The rules are well-founded by examples and, moreover, inconsistencies manifested in the examples are neither corrected nor aggregated by a global function or relation. The rough set approach does not need any additional information like probability in statistics or grade of membership in fuzzy set theory. It is conceptually simple and needs simple algorithms.

The set of rules can be used to support decisions concerning new coming objects. The decision support is made by matching a new object to one of rules; if it fails, a set of the 'nearest' rules is presented to the DM. In this paper, in order to find the 'nearest' rules, a new distance measure based on a valued closeness relation has been proposed. It involves concordance and discordance tests for threshold comparisons of the new object with each particular rule.

References

Fibak, J., Pawlak, Z., Slowinski, K., Slowinski, R. (1986). Rough sets based decision algorithm for treatment of duodenal ulcer by HSV. *Bulletin of the Polish Academy of Sciences, ser. Biological Sciences*, **34**, 227-246.

Grzymala-Busse, J.W. (1992). LERS - a system for learning from examples based on rough sets. In: Slowinski, ed. (1992), pp.3-18.

Krysinski, J. (1990). Rough sets approach to the analysis of the structure-activity relationship of quaternary imidazolium compounds. *Arzneimittel-Forschung/Drug Research* **40** (II), 795-799.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In: *Machine Learning* (R.S.Michalski, J.G.Carbonell, T.M.Mitchell, eds.), Morgan Kaufmann, pp.83-134.

Moscarola, J. (1978). Multicriteria decision aid - two applications in education management. In: *Multiple Criteria Problem Solving* (S.Zionts, ed.), Lecture Notes in Economics and Mathematical Systems, vol.155, Springer-Verlag, Berlin, pp.402-423.

Nowicki, R., Slowinski, R., Stefanowski, J. (1992). Rough sets analysis of diagnostic capacity of vibroacoustic symptoms. *Journal of Computers & Mathematics with Applications* **24**, 109-123.

Pawlak, Z. (1982). Rough Sets. *International Journal of Information & Computer Sciences* **11**, 341-356.

Pawlak, Z. (1991). *Rough Sets. Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht/Boston/ London.

Roubens, M., Vincke, Ph. (1985). *Preference Modelling*. Lecture Notes in Economics and Mathematical Systems, vol. 250, Springer-Verlag, Berlin.

Roy, B. (1985). *Méthodologie Multicritère d'Aide à la Décision*. Economica, Paris.

Roy, B. (1989). Main sources of inaccurate determination, uncertainty and imprecision in decision models. *Math. Comput. Modell.*, **12**, 1245-1254.

Roy, B. (1992). Decision science or decision aid science. *European Journal of Operational Research*, Special Issue on Model Validation in Operations Research (to appear).

Slowinski, R., ed. (1992). *Intelligent Decision Support. Applica tions and Advances of the Rough Sets Theory*. Kluwer Academic Publishers, Dordrecht/Boston/London.

Slowinski, K., Slowinski, R., Stefanowski, J. (1988). Rough sets approach to analysis of data from peritoneal lavage in acute pancreatitis. *Medical Informatics* **13**, 143-159.

Slowinski, R., Stefanowski, J. (1992). 'RoughDAS' and 'RoughClass' software implementations of the rough sets approach. In: Slowinski, ed. (1992), pp. 445-456.

Stefanowski, J. (1992). Classification support based on the rough sets theory. *Proc. IIASA Workshop on User-Oriented Methodology and Techniques of Decision Analysis and Support, Serock, Sept. 9-13, 1991* (to be published by Springer-Verlag).

Authors Index

Agusti, J. 245
Ambroszkiewicz, S. 518
Bagai, R. 415
Baroglio, C. 425
Bateman, M. 450
Berlandier, P. 375
Bittencourt, G. 538
Bosc, P. 209
Brown Jr., A.L. 362
Bry, F. 116
Busch, D.R. 29
Caselli, S. 496
Chadha, R. 255
Charlton, P. 612
Chau, C.W.R. 306
Chen, J. 152
Chou, S.C. 415
Chu, B.-T.B. 591
Chu, H. 19
De Raedt, L. 435
Dewan, H.M. 186
Di Manzo, M. 548
Du, H. 591
Eloff, J.H.P. 106
Fraňová, M. 476
Gaasterland, T. 198
Gan, H. 466
Giordana, A. 425
Giordano, L. 59
Giunchiglia, E. 548
Gross, M. 476
Grosz, G. 486
Hähnle, R. 49
Hesketh, J. 245
Imam, I.F. 395
Iwatani, T. 285
Klauck, C. 571
Klut, J.P. 106
Kodratoff, Y. 476
Lambrix, P. 162
Lavrač, N. 435
Lee, S.-J. 76
Leone, N. 235
Levy, J. 245
Liau, C.J. 316
Lietard, L. 209
Lin, B.I-P. 316
Lingras, P. 306
Ljung, L. 338
Lobo, J. 198
López, B. 96
Lounis, H. 405
Lowry, M.R. 219
Mantha, S. 362
Marek, V.W. 142
Martin, S. 450
Meyer, M.A. 385
Michalski, R.S. 395
Minker, J. 1
Müller, J.P. 385
Murray, N.V. 275
Natali, A. 496
Nebel, B. 132
Okamoto, W. 285
Orman, L.V. 172
Padgham, L. 132
Palopoli, L. 235
Pivert, O. 209
Plaisted, D.A. 19, 255
Plaza, E. 96
Posegga, J. 39
Puget, J.-F. 350
Rajasekar, A. 265
Rasiowa, H. 142
Rauszer, C.M. 326
Robertson, D. 245
Rolland, C. 486
Romeo, M. 235
Rönnquist, R. 162
Rosenthal, E. 275
Ruiz, C. 1
Saitta, L. 425
Sandewall, E. 558
Schaerf, A. 508
Schwagereit, J. 571
Shanbhogue, V. 415
Shue, L.-Y. 69
Skowron, A. 295
Slade, A. 450
Słowiński, R. 642
Sobolewski, M. 601
Stolfo, S.J. 186
Tano, S. 285
Thirunarayan, K. 528
Valiente, G. 86
Wakayama, T. 362
Wang, K. 581
Wong, S.K.M. 306
Wu, C.-H. 76
Wüthrich, B. 622
Zamani, R. 69
Zanichelli, F. 496
Zembowicz, R. 632
Żytkow, J.M. 415, 632

Springer-Verlag and the Environment

We at Springer-Verlag firmly believe that an international science publisher has a special obligation to the environment, and our corporate policies consistently reflect this conviction.

We also expect our business partners – paper mills, printers, packaging manufacturers, etc. – to commit themselves to using environmentally friendly materials and production processes.

The paper in this book is made from low- or no-chlorine pulp and is acid free, in conformance with international standards for paper permanency.

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 513: N. M. Mattos, An Approach to Knowledge Base Management. IX, 247 pages. 1991.

Vol. 515: J. P. Martins, M. Reinfrank (Eds.), Truth Maintenance Systems. Proceedings, 1990. VII, 177 pages. 1991.

Vol. 517: K. Nökel, Temporally Distributed Symptoms in Technical Diagnosis. IX, 164 pages. 1991.

Vol. 518: J. G. Williams, Instantiation Theory. VIII, 133 pages. 1991.

Vol. 522: J. Hertzberg (Ed.), European Workshop on Planning. Proceedings, 1991. VII, 121 pages. 1991.

Vol. 535: P. Jorrand, J. Kelemen (Eds.), Fundamentals of Artificial Intelligence Research. Proceedings, 1991. VIII, 255 pages. 1991.

Vol. 541: P. Barahona, L. Moniz Pereira, A. Porto (Eds.), EPIA '91. Proceedings, 1991. VIII, 292 pages. 1991.

Vol. 542: Z. W. Ras, M. Zemankova (Eds.), Methodologies for Intelligent Systems. Proceedings, 1991. X, 644 pages. 1991.

Vol. 543: J. Dix, K. P. Jantke, P. H. Schmitt (Eds.), Nonmonotonic and Inductive Logic. Proceedings, 1990. X, 243 pages. 1991.

Vol. 546: O. Herzog, C.-R. Rollinger (Eds.), Text Understanding in LILOG. XI, 738 pages. 1991.

Vol. 549: E. Ardizzone, S. Gaglio, F. Sorbello (Eds.), Trends in Artificial Intelligence. Proceedings, 1991. XIV, 479 pages. 1991.

Vol. 565: J. D. Becker, I. Eisele, F. W. Mündemann (Eds.), Parallelism, Learning, Evolution. Proceedings, 1989. VIII, 525 pages. 1991.

Vol. 567: H. Boley, M. M. Richter (Eds.), Processing Declarative Kowledge. Proceedings, 1991. XII, 427 pages. 1991.

Vol. 568: H.-J. Bürckert, A Resolution Principle for a Logic with Restricted Quantifiers. X, 116 pages. 1991.

Vol. 587: R. Dale, E. Hovy, D. Rösner, O. Stock (Eds.), Aspects of Automated Natural Language Generation. Proceedings, 1992. VIII, 311 pages. 1992.

Vol. 590: B. Fronhöfer, G. Wrightson (Eds.), Parallelization in Inference Systems. Proceedings, 1990. VIII, 372 pages. 1992.

Vol. 592: A. Voronkov (Ed.), Logic Programming. Proceedings, 1991. IX, 514 pages. 1992.

Vol. 596: L.-H. Eriksson, L. Hallnäs, P. Schroeder-Heister (Eds.), Extensions of Logic Programming. Proceedings, 1991. VII, 369 pages. 1992.

Vol. 597: H. W. Guesgen, J. Hertzberg, A Perspective of Constraint-Based Reasoning. VIII, 123 pages. 1992.

Vol. 599: Th. Wetter, K.-D. Althoff, J. Boose, B. R. Gaines, M. Linster, F. Schmalhofer (Eds.), Current Developments in Knowledge Acquisition - EKAW '92. Proceedings. XIII, 444 pages. 1992.

Vol. 604: F. Belli, F. J. Radermacher (Eds.), Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. Proceedings, 1992. XV, 702 pages. 1992.

Vol. 607: D. Kapur (Ed.), Automated Deduction - CADE-11. Proceedings, 1992. XV, 793 pages. 1992.

Vol. 610: F. von Martial, Coordinating Plans of Autonomous Agents. XII, 246 pages. 1992.

Vol. 611: M. P. Papazoglou, J. Zeleznikow (Eds.), The Next Generation of Information Systems: From Data to Knowledge. VIII, 310 pages. 1992.

Vol. 617: V. Mařík, O. Štěpánková, R. Trappl (Eds.), Advanced Topics in Artificial Intelligence. Proceedings, 1992. IX, 484 pages. 1992.

Vol. 619: D. Pearce, H. Wansing (Eds.), Nonclassical Logics and Information Processing. Proceedings, 1990. VII, 171 pages. 1992.

Vol. 622: F. Schmalhofer, G. Strube, Th. Wetter (Eds.), Contemporary Knowledge Engineering and Cognition. Proceedings, 1991. XII, 258 pages. 1992.

Vol. 624: A. Voronkov (Ed.), Logic Programming and Automated Reasoning. Proceedings, 1992. XIV, 509 pages. 1992.

Vol. 627: J. Pustejovsky, S. Bergler (Eds.), Lexical Semantics and Knowledge Representation. Proceedings, 1991. XII, 381 pages. 1992.

Vol. 633: D. Pearce, G. Wagner (Eds.), Logics in AI. Proceedings. VIII, 410 pages. 1992.

Vol. 636: G. Comyn, N. E. Fuchs, M. J. Ratcliffe (Eds.), Logic Programming in Action. Proceedings, 1992. X, 324 pages. 1992.

Vol. 638: A. F. Rocha, Neural Nets. A Theory for Brains and Machines. XV, 393 pages. 1992.

Vol. 642: K. P. Jantke (Ed.), Analogical and Inductive Inference. Proceedings 1992. VIII, 319 pages. 1992.

Vol. 659: G. Brewka, K. P. Jantke, P. H. Schmitt (Eds.), Nonmonotonic and Inductive Logic. Proceedings, 1991. VIII, 332 pages. 1993.

Vol. 660: E. Lamma, P. Mello (Eds.), Extensions of Logic Programming. Proceedings, 1992. VIII, 417 pages. 1993.

Vol. 667: P. B. Brazdil (Ed.), Machine Learning: ECML - 93. Proceedings, 1993. XII, 471 pages. 1993.

Vol. 671: H. J. Ohlbach (Ed.), GWAI-92: Advances in Artificial Intelligence. Proceedings, 1992. XI, 397 pages. 1993.

Vol. 689: J. Komorowski, Z. W. Raś (Eds.), Methodologies for Intelligent Systems. Proceedings, 1993. XI, 653 pages. 1993.

Lecture Notes in Computer Science

Vol. 651: R. Koymans, Specifying Message Passing and Time-Critical Systems with Temporal Logic. IX, 164 pages. 1992.

Vol. 652: R. Shyamasundar (Ed.), Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1992. XIII, 405 pages. 1992.

Vol. 653: A. Bensoussan, J.-P. Verjus (Eds.), Future Tendencies in Computer Science, Control and Applied Mathematics. Proceedings, 1992. XV, 371 pages. 1992.

Vol. 654: A. Nakamura, M. Nivat, A. Saoudi, P. S. P. Wang, K. Inoue (Eds.), Prallel Image Analysis. Proceedings, 1992. VIII, 312 pages. 1992.

Vol. 655: M. Bidoit, C. Choppy (Eds.), Recent Trends in Data Type Specification. X, 344 pages. 1993.

Vol. 656: M. Rusinowitch, J. L. Rémy (Eds.), Conditional Term Rewriting Systems. Proceedings, 1992. XI, 501 pages. 1993.

Vol. 657: E. W. Mayr (Ed.), Graph-Theoretic Concepts in Computer Science. Proceedings, 1992. VIII, 350 pages. 1993.

Vol. 658: R. A. Rueppel (Ed.), Advances in Cryptology – EUROCRYPT '92. Proceedings, 1992. X, 493 pages. 1993.

Vol. 659: G. Brewka, K. P. Jantke, P. H. Schmitt (Eds.), Nonmonotonic and Inductive Logic. Proceedings, 1991. VIII, 332 pages. 1993. (Subseries LNAI).

Vol. 660: E. Lamma, P. Mello (Eds.), Extensions of Logic Programming. Proceedings, 1992. VIII, 417 pages. 1993. (Subseries LNAI).

Vol. 661: S. J. Hanson, W. Remmele, R. L. Rivest (Eds.), Machine Learning: From Theory to Applications. VIII, 271 pages. 1993.

Vol. 662: M. Nitzberg, D. Mumford, T. Shiota, Filtering, Segmentation and Depth. VIII, 143 pages. 1993.

Vol. 663: G. v. Bochmann, D. K. Probst (Eds.), Computer Aided Verification. Proceedings, 1992. IX, 422 pages. 1993.

Vol. 664: M. Bezem, J. F. Groote (Eds.), Typed Lambda Calculi and Applications. Proceedings, 1993. VIII, 433 pages. 1993.

Vol. 665: P. Enjalbert, A. Finkel, K. W. Wagner (Eds.), STACS 93. Proceedings, 1993. XIV, 724 pages. 1993.

Vol. 666: J. W. de Bakker, W.-P. de Roever, G. Rozenberg (Eds.), Semantics: Foundations and Applications. Proceedings, 1992. VIII, 659 pages. 1993.

Vol. 667: P. B. Brazdil (Ed.), Machine Learning: ECML – 93. Proceedings, 1993. XII, 471 pages. 1993. (Subseries LNAI).

Vol. 668: M.-C. Gaudel, J.-P. Jouannaud (Eds.), TAPSOFT '93: Theory and Practice of Software Development. Proceedings, 1993. XII, 762 pages. 1993.

Vol. 669: R. S. Bird, C. C. Morgan, J. C. P. Woodcock (Eds.), Mathematics of Program Construction. Proceedings, 1992. VIII, 378 pages. 1993.

Vol. 670: J. C. P. Woodcock, P. G. Larsen (Eds.), FME '93: Industrial-Strength Formal Methods. Proceedings, 1993. XI, 689 pages. 1993.

Vol. 671: H. J. Ohlbach (Ed.), GWAI-92: Advances in Artificial Intelligence. Proceedings, 1992. XI, 397 pages. 1993. (Subseries LNAI).

Vol. 672: A. Barak, S. Guday, R. G. Wheeler, The MOSIX Distributed Operating System X, 221 pages. 1993.

Vol. 673: G. Cohen, T. Mora, O. Moreno (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings, 1993. X, 355 pages 1993.

Vol. 674: G. Rozenberg (Ed.), Advances in Petri Nets 1993 VII, 457 pages. 1993.

Vol. 675: A. Mulkers, Live Data Structures in Logic Programs. VIII, 220 pages. 1993.

Vol. 676: Th. H. Reiss, Recognizing Planar Objects Using Invariant Image Features. X, 180 pages. 1993.

Vol. 677: H. Abdulrab, J.-P. Pécuchet (Eds.), Word Equations and Related Topics, Proceedings, 1991. VII, 214 pages. 1993.

Vol. 678: F. Meyer auf der Heide, B. Monien, A. L. Rosenberg (Eds.), Parallel Architectures and Their Efficient Use. Proceedings, 1992. XII, 227 pages. 1993.

Vol. 683: G.J. Milne, L. Pierre (Eds.), Correct Hardware Design and Verification Methods. Proceedings, 1993. VIII, 270 Pages. 1993.

Vol. 684: A. Apostolico, M. Crochemore, Z. Galil, U. Manber (Eds.), Combinatorial Pattern Matching. Proceedings, 1993. VIII, 265 pages. 1993.

Vol. 685: C. Rolland, F. Bodart, C. Cauvet (Eds.), Advanced Information Systems Engineering. Proceedings, 1993. XI, 650 pages. 1993.

Vol. 686: J. Mira, J. Cabestany, A. Prieto (Eds.), New Trends in Neural Computation. Proceedings, 1993. XII, 746 pages. 1993.

Vol. 687: H. H. Barrett, A. F. Gmitro (Eds.), Information Processing in Medical Imaging. Proceedings, 1993. XVI, 567 pages. 1993.

Vol. 688: M. Gauthier (Ed.), Ada: Without Frontiers. Proceedings, 1993. VIII, 353 pages. 1993.

Vol. 689: J. Komorowski, Z. W. Raś (Eds.), Methodologies for Intelligent Systems. Proceedings, 1993. XI, 653 pages. 1993. (Subseries LNAI).