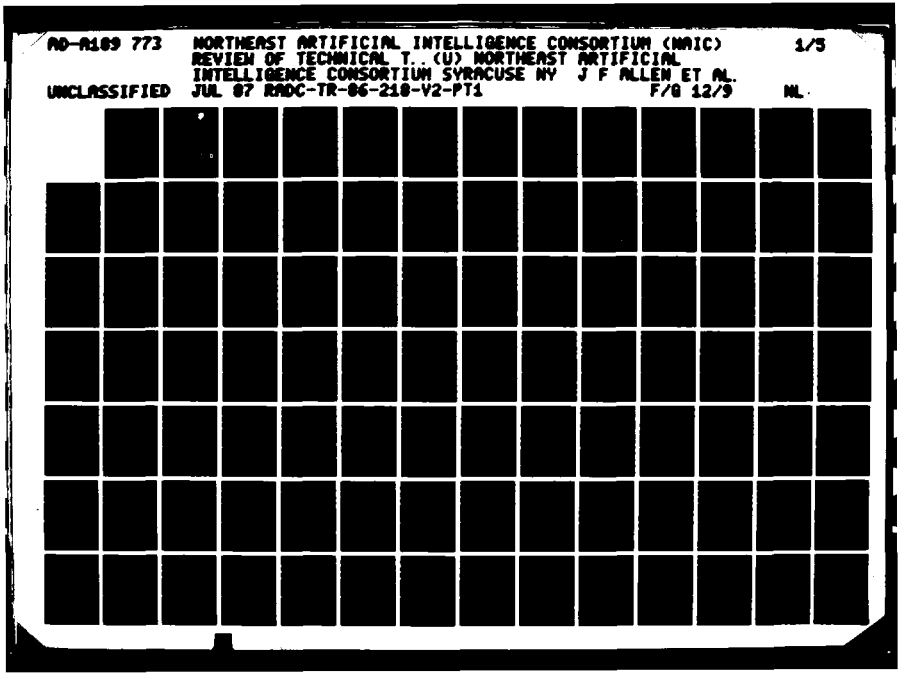
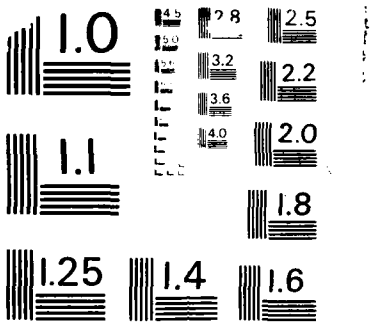

AD-A189 773 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC) 1/5
REVIEW OF TECHNICAL T.. (U) NORTHEAST ARTIFICIAL
INTELLIGENCE CONSORTIUM SYRACUSE NY J F ALLEN ET AL.
UNCLASSIFIED JUL 87 RADC-TR-86-210-V2-PT1 F/G 12/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963

DTIC FILE COPY

AD-A189 773

RADC-TR-86-218, Vol II, Part I (of two)
Interim Report
July 1987

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC)
Review of Technical Tasks

Syracuse University

J. F. Allen, P. B. Berra, J. A. Biles, K. A. Bowen, S. E. Conry, W. B. Croft, V. R. Lesser, R. A. Meyer, J. W. Modestino, G. Nagy, S. Nirenburg, H. Rhody, J. E. Searleman, S. C. Shapiro, S. N. Srihari and B. Woolf

This effort was partially funded by the Laboratory Directors' Fund

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC ELECTE DEC 3 1 1987 S D H

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

87 12 21 031

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

Because of the size of this volume, it has been separated into Volume II, Part 1 (pages 1 - 466) and Volume II, Part 2 (pages 467 - 936).

RADC-TR-86-218, Volume II, Part 1 (of two) has been reviewed and is approved for publication.

APPROVED: Donald J. Gondek

DONALD J. GONDEK
NAIC Program Manager

APPROVED: Raymond P. Urtz

RAYMOND P. URTZ, Jr.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Richard W. Poulliot

RICHARD W. POULIOT
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS N/A
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5 MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-86-218, Volume II, Part 1 (of two)
6a NAME OF PERFORMING ORGANIZATION Syracuse University ATTN: Dr. Volker Weiss, Director 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c ADDRESS (City, State, and ZIP Code) Northeast Artificial Intelligence Consortium 120 Hinds Hall Syracuse NY 13210	7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b OFFICE SYMBOL (If applicable) COES	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F (over) PROJECT NO: 2304 TASK NO: J5 WORK UNIT ACCESSION NO: 01

11 TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC)
Review of Technical Tasks

12 PERSONAL AUTHOR(S) (See reverse)
J.F. Allen, P.B. Berra, J.A. Biles, K.A. Bowen, S.E. Conry, W.B. Croft, V.R. Lesser

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM Dec 84 TO Dec 85	July 1987	478

16 SUPPLEMENTARY NOTATION
This effort was partially funded by the Laboratory Directors' Fund

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Expert System, Distributed AI, Communications System Control,
12	05		Image Interpretation, AI Planning, Knowledge-based Reasoning,
12	07		Knowledge Acquisition, Belief Revision, (See reverse)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the first year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photo interpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Donald J. Gondek	22b TELEPHONE (Include Area Code) (315) 330-7794 22c OFFICE SYMBOL RADC (COES)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

UNCLASSIFIED

Block 10 (Cont'd) Program Element No.	Project No.	Task No.	Work Unit Accession No.
61101F, 61102F,	5581	27	13
33126F	4594	18	E2
	2155	02	10
	LDFP	15	C4

Block 12 (Cont'd)

R.A. Meyer, J.W. Modestino, G. Nagy, S. Nirenburg, H. Rhody, J.E. Searleman, S.C. Shapiro, S.N. Srihari and B. Woolf

Block 16 (Cont'd)

Because of the size of this volume, it has been separated into Volume II, Part 1 (pages 1 - 466) and Volume II, Part 2 (pages 467 - 936).

Block 18 (Cont'd)

Domain Knowledge, Service Restoral, Poplar, Procedural and Semantic Data Bases, Plan Recognition, Language Parsing, Interval-based Theory of Time, SNePS, Speech Understanding, AI Architecture, Knowledge-based Maintenance, Knowledge Explanation System, ThinkerToy

Acknowledgements

The authors wish to thank Dr. Northrup Fowler, Rome Air Development Center (RADC), who conceived the Northeast Artificial Intelligence Consortium (NAIC), Mr. Jake Scherer of RADC, who was instrumental in implementing the NAIC, and Mr. Donald Gondek, who, as Project Manager, rendered valuable assistance and communicated the needs of RADC. The authors are most appreciative of the encouragement, support, and friendly suggestions made by Mr. Fred I. Diamond, Chief Scientist of RADC, and Mr. Ray P. Urtz, Technical Director of the Command and Control Division of RADC. Special thanks are due to Dr. Bradley J. Strait and Mrs. Andrea Pflug of Syracuse University and to Mrs. Estella G. Bray of Clarkson University for their work in making the NAIC a success and in completing this report.

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Table of Contents

Vol. II, Part 1

1 INTRODUCTION 1

1.1 The Northeast Artificial Intelligence Consortium 1

1.2 The Topics Under Study and the Principal Investigators (P.I.s) at Each Institution 1

2 VMES: A NETWORK-BASED VERSATILE MAINTENANCE EXPERT SYSTEM (SUNY Buffalo) 4

2.1 Device Modeling and Fault Diagnosis of VMES 5

2.2 Graphical Interface of VMES 18

References 26

Appendix 2-A: A Fault Diagnosis System Based on an Integrated Knowledge Base 27
Appendix 2-B: A Logic for Belief Revision 31
Appendix 2-C: A Model for Belief Revision 91
Appendix 2-D: SNePS Considered as a Fully Intensional Propositional Semantic Network 179
Appendix 2-E: Belief Revision in SNePS 227

3 DISTRIBUTED PROBLEM SOLVING (Clarkson University) 245

3.1 Introduction 246

3.2 Application Domain Description 247

3.3 System Architecture 251

3.4 Knowledge Representation 254

3.5 Problem Solving Strategies 260

3.6 Progress Review and Future Directions 264

Bibliography 265

Appendix 3-A Shared Knowledge Base for Independent Problem Solving Agents 267

4 PLANNER SYSTEM FOR THE APPLICATION OF INDICATIONS AND WARNING
(Colgate University) 277

4.1 Introduction 278

4.2 Strategy 278

4.3 POPLAR 1.3 279

4.4 POPLAR 2.0 294

4.5 Background and Related Work 303

4.6 Status and Future Work 306

Bibliograghy 308

Appendix 1. Representation of Objects in POPLAR 1.3 311
Appendix 2. Examples of PLAN representation in POPLAR 1.3 315
Appendix 3. Examples of POPLAR 1.3 rating functions 319
Appendix 4. HISTORY in POPLAR 1.3 321
Appendix 5. BLACKBOARD in POPLAR 1.3 321
Appendix 6. Examples of PLAN reprentation in POPLAR 2.0 323
Appendix 7. Providing Intelligent Assistance in Distributed
Office Environments 327
Appendix 8. POPLAR: A Testbed for Cognitive Modeling 345

5 PLAN RECOGNITION, KNOWLEDGE ACQUISITION AND EXPLANATION IN AN
INTELLIGENT INTERFACE
(University of Massachusetts) 375

5.1 Basic Issues 378

5.2 Plan Recognition/Planning 379

5.3 Interface Tools 382

5.4 Discourse Processing in a Knowledge Acquistion Interface 389

5.5 Understanding Discourse Conventions in Tutoring 403

5.6 A Knowledge-Based Approach to Data Management
for Intelligent User Interfaces 431

5.7 POISE Graphical Interface for Task Specification..... 453

Table of Contents

Vol. II, Part 2

6 AUTOMATIC PHOTO INTERPRETATION
(Rensselaer Polytechnic Institute) 467

6.1 Introduction 468

6.2 Transformation Invariant Attributes for
Digitized Object Outlines 469

6.3 Design of an Inference Engine for an
Image Interpretation Expert System 472

6.4 Current plans 480

Attachment A: Design of an Inference Engine for an Image
Interpretation Expert System - IPL-TR-067 485
Attachment B: Transformation Invariant Attributes for
Digital Object Outlines 597

7 SPEECH UNDERSTANDING RESEARCH
(Rochester Institute of Technology) 701

7.1 Introduction 702

7.2 Research Activities 703

7.3 Speech Scientist's Workbench 708

7.4 Tasks for the Coming Year..... 710

Bibliography 712

8 TIME-ORIENTED PROBLEM SOLVING
(The University of Rochester) 715

8.1 Introduction 716

8.2 Description of Research Accomplished 716

8.3 Future Research 717

Appendix 8-A: A Common Sense Theory of Time 719
Appendix 8-B: A Model of Naive Temporal Reasoning 731
Appendix 8-C: Toward a Theory of Plan Recognition 749
Appendix 8-D: A Formal Logic that Supports Planning with a Partial
Description of the Future 817

9 COMPUTER ARCHITECTURES FOR VERY LARGE KNOWLEDGE BASES
(Syracuse University) 835

9.1 Introduction 836

9.2 Partial Match Retrieval 838

9.3 Surrogate Files 840

9.4 Computer Architecture for partial Match Retrievals 845

9.5 Summary 847

9.6 Future Plans 850

References 852

and

KNOWLEDGE BASE MAINTENANCE USING LOGIC PROGRAMMING METHODOLOGIES
(Syracuse University) 853

9.7 Background and Goals 855

9.8 Related concerns 857

9.9 Future plans 858

Appendix 9-A: Meta-Level Programming and Knowledge Representation . 859
Appendix 8-B: Fast Decompilation of Compiled Prolog Clauses 893
Appendix 8-C: The Design and Implementation of a High-Speed
Incremental Portable Prolog Compiler.. 901
Appendix 8-D: Compiler Optimizations for the WAM 915
Appendix 8-E: A Meta-Level Extension of Prolog 923

1 INTRODUCTION

1.1 The Northeast Artificial Intelligence Consortium

The Northeast Artificial Intelligence Consortium (NAIC) is a group of eight institutions of higher learning organized for the purpose of developing research and education in Artificial Intelligence (AI). The participating institutions are:

State University of New York at Buffalo, Buffalo, New York

Clarkson University, Potsdam, New York

Colgate University, Hamilton, New York

The University of Massachusetts at Amherst, Amherst, Massachusetts

Rensselaer Polytechnic Institute, Troy, New York

Rochester Institute of Technology, Rochester, New York

The University of Rochester, Rochester, New York

Syracuse University, Syracuse, New York.

Researchers at each institution have their own expertise and interests and are addressing a varied group of problems in AI that are of interest to the Air Force. Each of these problems has been viewed as a more or less distinct task and as such each research group has submitted a complete report covering the research task(s) undertaken at its institution during the last year. Therefore the task(s) at each institution is reviewed in a separate section of this report. The first page of each of these sections is a detailed table of contents for that section. A more general outline of each research section is included in the main table of contents. A list of references or a bibliography is included in each section as needed. Research papers, where available, to substantiate these reviews are included in the body of the review or as attachments at the end of each section. These things were done to aid the reader in the understanding of each section.

1.2 The Topics Under Study and the Principal Investigators (P.I.s) at Each Institution:

A) VMES: A NETWORK-BASED VERSATILE MAINTENANCE EXPERT SYSTEM

P.I.s: Stuart C. Shapiro and Sargur N. Srihari
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260

Report submitted by:
Stuart C. Shapiro

Sargur N. Srihari
James Geller, Graduate Research Assistant
Ming-Ruey Taie, Graduate Research Assistant
Scott S. Campbell, Graduate Research Assistant

B) DISTRIBUTED PROBLEM SOLVING

P.I.s: Susan E. Conry, and Robert A Meyer
Electrical and Computer Engineering
Clarkson University
Potsdam, NY 13676
and
Janice E. Searleman
Mathematics and Computer Science
Clarkson University
Potsdam, NY 13676

Report submitted by:
Susan E. Conry
Robert A. Meyer
Janice E. Searleman

C) PLANNER SYSTEM FOR THE APPLICATION OF INDICATIONS AND WARNING

P.I.: Sergei Nirenburg
Computer Science Department
Colgate University
Hamilton, NY 13346

Report submitted by:
Sergei Nirenburg

D) PLAN RECOGNITION, KNOWLEDGE ACQUISITION AND EXPLANATION IN AN INTELLIGENT INTERFACE

P.I.s: Victor Lesser, W. Bruce Croft and Beverly Woolf
Department of Computer and Information Science
The University of Massachusetts
Amherst, MA 01003

Report submitted by:
Victor Lesser
W. Bruce Croft
Beverly Woolf

E) AUTOMATIC PHOTO INTERPRETATION

P.I.s: J. W. Modestino and G. Nagy
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

Report submitted by:
J. W. Modestino
G. Nagy

F) SPEECH UNDERSTANDING RESEARCH

P.I.: Harvey Rhody
RIT Research Corporation
75 Highpower Road
Rochester, NY 14623
and
John A. Biles
Computer Science Department
Rochester Institute of Technology
Rochester, NY

Report submitted by:
Harvey Rhody

G) TIME-ORIENTED PROBLEM SOLVING

P.I.: James F. Allen
Computer Science Department
The University of Rochester
Rochester, NY 14627

Report submitted by:
James F. Allen

H 1) COMPUTER ARCHITECTURES FOR VERY LARGE KNOWLEDGE BASES

P.I.: P. Bruce Berra
Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13210

Report submitted by:
P. Bruce Berra

2) KNOWLEDGE BASE MAINTENANCE USING LOGIC PROGRAMMING METHODOLOGIES

P.I.: Kenneth A. Bowen
School of Computer and Information Science
Syracuse University
Syracuse, NY 13210

Report submitted by:
Kenneth Bowen

2 VMES: A NETWORK-BASED VERSATILE MAINTENANCE EXPERT SYSTEM

Report submitted by:
Staurt C. Shapiro
Sargur N. Srihari
James Geller, Graduate Research Assistant
Ming-Ruey Taie, Graduate Research Assistant
Scott S. Campbell, Graduate Research Assistant

Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260

TABLE OF CONTENTS

Part I Device Modeling and Fault Diagnosis of VMES 5
1 Introduction 5
2 Version 1 5
3 Version 2 7
3.1 Device Representation 7
3.1.1 Structural Description 7
3.1.2 Functional Description 10
3.2 Inference Engine 11
3.3 Demonstration Example 12
4 Discussion 16
5 Future Work 17

Part II Graphical Interface of VMES 18
1 Introduction 18
2 The Graphical Interface 18
2.1 Motivation 18
2.2 Components of the interface 19
2.3 How display works 19
2.4 Special display Parameters 20
2.4.1 Modality 20
2.4.2 Pruning the display 20
2.4.3 Optimal screen use 22
2.5 Graphical Inference Trace 23
3 Future Work 25

References 26

Appendix 2-A: A Fault Diagnosis System Based on an Integrated Knowledge Base 27
Appendix 2-B: A Logic for Belief Revision 31
Appendix 2-C: A Model for Belief Revision 91
Appendix 2-D: SNePS Considered as a Fully Intensional Propositional Semantic Network 179
Appendix 2-E: Belief Revision in SNePS 227

Part. I Device Modeling and Fault Diagnosis of VMES

1. INTRODUCTION

We are developing a versatile maintenance expert system (VMES) for trouble-shooting digital circuits.

Some diagnosis systems, such as MYCIN [19] for medical diagnosis and CRIB [5] for computer fault diagnosis, are built on rules which represented empirical associations. Though these systems have had considerable success, there are some important drawbacks: knowledge acquisition from domain experts is difficult, all possible faults (diseases) have to be enumerated explicitly which results in a limitation of the diagnosis power, and they have almost no capability of system generalization.

As a solution to difficulties of empirical-rule-based diagnosis systems, structural and functional descriptions have been widely used by AI researchers in the domain of fault diagnosis [3, 4]. The knowledge needed for building such a system is well-structured and readily available at the time when a device is designed. There is no need to explicitly enumerate all possible faults since they are defined generically as violated expectations at the output ports. This approach makes the adaptation for the system to a new device much easier, because the only thing needed is to describe the device to the system.

To test this idea, we have implemented a diagnosis system that has successfully pinpointed the faulty part of a multiplier/adder board, a favorite example for researchers in this field (see e.g. [3].)

An important aspect of our research is to find a good knowledge-representation scheme to support the diagnosis and the construction of a versatile maintenance system. We have implemented our system in the SNePS Semantic Network Processing System [16]. Advantages are: (1) structural and functional knowledge is integrated into a single network; (2) reasoning is done by rule-based inference; (3) diagnosis assumptions can be handled in a natural way; (4) the deduction process can be monitored; (5) inference can also be traced graphically; (6) the representation can easily be expanded and modified; (7) procedural knowledge is represented and used; (8) it is smoothly interfaced with LISP.

Version 1 of our implementation uses a hand-coded description of the device. An intermediate user, who adapts the system to a specific device, needs to hand-code all the structural and functional details of the device, even that a lot of parts are of the same component type. Since versatility is a goal, the system was redesigned as Version 2. It contains a kind of type declaration to build a component library. This has enhanced the versatility of the system quite significantly. We successfully adapted the system to a new device with minimum effort by just adding the descriptions of new components to the system.

A brief description of the implementation of Version 1 appears in the next section. Section 3 contains a detailed description of our current implementation (Version 2) along with an annotated demonstration. Section 4 and 5 are discussions and future work.

2. VERSION 1

This section contains a brief description of our early implementation of VMES. A board of three multipliers and two adders was used as the target object to be diagnosed.

The structural description was hierarchical, which made it possible to focus on the relevant part of the device at any time during the diagnosis. The structural description was hard-wired, every detail of the device needed to be entered by hand. Examples are:

In SNePS codes			*In English*
(build	object	D1	The object D1 is of M3A2 type;
	type	M3A2	it has three inputs and two outputs
	inp1	D1inp1	named in order as D1inp1, D1inp2,
	inp2	D1inp2	D1inp3, D1out1 and D1out2;
	inp3	D1inp3	it is consists of 5 sub-parts: D1M1,
	out1	D1out1	D1M2, D1M3, D1A1 and D1A2.
	out2	D1out2	
	sub-part	(D1M1 D1M2 D1M3 D1A1 D1A2)	
	super-part	NIL)	
(build	object	D1M1	The object D1M1 is a MULTiplier,
	type	MULT	it has two inputs and one output
	inp1	D1M1inp1	named in order as D1inp1, D1inp2,
	inp2	D1M1inp2	and D1out1. It has no sub-part and
	out1	D1M1out1	it is part of D1.
	sub-part	NIL	
	super-part	D1)	
(build	from	D1inp1	There is a wire connection from
	to	(D1M1inp1 D1M2inp1))	D1inp1 to D1M1inp1 & D1M2inp1.

Only one wire connection description, which actually represented two wires, is shown here. And similar codes for D1M2, D1M3, D1A1 and D1A2 were required since the structural description was hand-coded.

Functional definition was implemented as a template of the SNePSUL function node [17]. Unlike the structural description, the functional description was associated with each type of the components rather than the parts themselves. An example was:

In SNePS code:

```
(dp ADDER (inp1 inp2 out1)
   (cond ((eq (plus inp1 inp2) out1) (succeed true))
         (t (succeed false))))
```

In English:

For an adder, it is good if the two inputs sum to the output,
it is bad otherwise.

The function description gave an explicit definition to decide whether a component was

malfunctioning. It did not explicitly depict what the function of an adder was, which was required in order to simulate the behavior of an adder. And for every type of component, it needed its own rule for finding violated expectation at the output ports. The rules, in English, looked like:

If an object is an adder and all its input and output values are known, then one and only one of the following is true:
1). the object is functioning well, which can be infered from the adder function description;
2). there is a violated expectation at its output.

The inference engine for fault diagnosis followed a simple control structure. It is similar to that of the current implementation, and is discussed in the next section.

3. VERSION 2

This section contains a full description of our current system implementation. The system consists of two major parts: the device representation and the inference engine. The device representation is further divided into structural and functional descriptions. An annotated demonstration of the system is at the end of this section.

3.1. Device Representation

The current implementation of VMES includes a complete redesign of the device representation in both the structural and functional description. The disadvantages of the hand-coded description have been removed, and major progress has been made toward an ultimately versatile system.

3.1.1. Structural Description

Once again, only the logical structure of the device is represented and used for diagnosis in our current implementation. Instead of hand-coding every detail of the device, the system keeps a component library which know every "type" of component. Each component type is abstracted at two levels and represented by two SNePS rules which are categorized as instantiation rules. The structure of the device is still represented in a hierarchical way through the parts hierarchy. Sub-parts of the device are instantiated only when they are needed. This increases memory efficiency.

At level-1 instantiation, an object is built as a module (a black box) with its I/O ports and a pointer to its functional description. The functional description is implemented as a LISP function which simulates/infers the value of one port in terms of the others. This will be discussed later.

At level-2, the sub-parts of the object at the next hierarchical level is built, and the wire connections between the object and its sub-parts, as well as those among the sub-parts themselves are made. Each sub-part is assigned a name which is an extension of the name of its super-part (the object), and it is instantiated at level-1 so that its I/O ports are available for the wire connections.

Several typical instantiation rules are as follows:

All annotations are shown in italics.

All types of components are described as:
level-1 description: I/O ports and functions.
level-2 description: sub-parts and connections.

```
; The following two SNePS rules describe
; the M3A2 type components:

(build
   avb $x
   ant (build object *x type M3A2 state TBI-L1)
   cq (build inport-of *x inp-id 1) = vINP1
   cq (build inport-of *x inp-id 2) = vINP2
   cq (build inport-of *x inp-id 3) = vINP3
   cq (build outport-of *x out-id 1) = vOUT1
   cq (build outport-of *x out-id 2) = vOUT2
   cq (build port *vOUT1 f-rule M3A2out1
            pn 3 p1 *vINP1 p2 *vINP2 p3 *vINP3)
   cq (build port *vOUT2 f-rule M3A2out2
            pn 3 p1 *vINP1 p2 *vINP2 p3 *vINP3]

; The first three lines says that "if x is an M3A2 and is to be
;   instantiated at level-1 (TBI-L1), then do the follows:"
; The next five lines instantiate the i/o ports.
; The last two "builds" link the output ports to the functional
;   description of the object. The first one says "to simulate the
;   value of first output, uses the function M3A2out1 which takes
;   three parameters: the inputs of the object x in order."
; Similar links can be done for all input ports if we want to infer
;   their values from other i/o ports.

(build
   avb *x
   ant (build object *x type M3A2 state TBI-L2)
   cq (build
      avb ($xp1 $xp2 $xp3 $xp4 $xp5)
      ant (build name: Give-PID-M3A2 object *x
               p1 *xp1 p2 *xp2 p3 *xp3 p4 *xp4 p5 *xp5)
      cq ((build object *xp1 type MULT state TBI-L1)
         (build object *xp2 type MULT state TBI-L1)
         (build object *xp3 type MULT state TBI-L1)
         (build object *xp4 type ADDER state TBI-L1)
         (build object *xp5 type ADDER state TBI-L1)
         (build super-part *x
              sub-parts (*xp1 *xp2 *xp3 *xp4 *xp5))
         (build from *vINP1
              to  ((build inport-of *xp1 inp-id 2)
                   (build inport-of *xp2 inp-id 1)))
         ; to save space, not all wire connections are shown here.
         (build from (build outport-of *xp3 out-id 1)
              to  (build inport-of *xp5 inp-id 2))
         (build from (build outport-of *xp5 out-id 1)
              to  *vOUT2]
```

```
; The first seven lines say: "if x is an M3A2 at TBI-L2, uses the
;    function Give-PID-M3A2 to get the names for its sub-parts"
; The next seven lines declare the types of the sub-parts and will
;    activate appropriate rules to instantiate them at their level-1
;    instantiation. The super-part/sub-parts hierarchical relation
;    between the object x and its sub-parts is built also.
; The remainders connect the wires between x and its sub-parts as
;    well as those among the sub-parts themselves.



; The following two SNePS rules describes
; the MULTiplier type components:

(build
   avb $x
   ant (build object *x type MULT state TBI-L1)
   cq (build inport-of *x inp-id 1) = vINP1
   cq (build inport-of *x inp-id 2) = vINP2
   cq (build outport-of *x out-id 1) = vOUT1
   cq (build port *vOUT1 f-rule MULTout1
            pn 2 p1 *vINP1 p2 *vINP2]

(build
   avb *x
   ant (build object *x type MULT state TBI-L2)
   cq (build super-part *x sub-parts lamDRU]

; Please note that the level-2 instantiation will not instantiate any
;    sub-part since a multiplier is regarded as a Depot Replaceable
;    Unit (DRU) and there is no need to represent its details.
```

All instantiation rules are stored in a file, which is regarded as a components library. Representing the structure of a device via the instantiation rules and the use of a components library give the system several important advantages. We do not have to hand-code three almost identical multipliers on our example digital circuit board; the information is generated by the system only when required during the course of diagnosis. This should minimize the construction effort during the system development period, and should also gain some memory efficiency during diagnosis. This is especially important in a memory critical environment.

Although instantiation during diagnosis is good for memory efficiency, it is slower during diagnosis. To overcome this problem without degrading the benefit of fast system construction, we designed the representation in a way which allows pre-instantiation of the device before diagnosis. This can be done easily by changing all TBI-L2 nodes in the components library to TBI-L1. Since the instantiation rules are used in a forward way, if a device is declared to be some type at its level-1 instantiation, it would activate all required instantiation rules throughout its structural hierarchies and build every detail of the device. This design gives the system one more dimension of versatility, namely that the system is versatile in both memory-critical and diagnosis-speed-critical situations.

The most important advantage of the current implementation is the extreme ease in adapting the system to other devices. All that the system adapter has to do is to add the structural and functional information of the "new" component types to the components library and the functions library, which will be discussed later. A new component type is defined as a component type which has not been described to the component library. The new device itself is a new component type by our definition. The effort required to adapt the system to new devices should be minimal since digital circuit devices have a lot of common components, and the structural and functional description should be readily available at the time when a device is designed.

3.1.2. Functional Description

In Version 1, the functional description was actually a testing procedure which could only be used to decide whether a component was malfunctioning. It had two main drawbacks: the description could not be used to simulate the behavior of the component, and every component type required its own associated SNePS rule for finding violated expectation at its output ports.

Version 1 offended a theoretical basis of fault diagnosis. It implemented the strategy:

> If the component is malfunctioning,
> there is violated expectation at its output.

But it should be the other way around:

> If some violated expectation is observed at the outputs,
> the component is malfunctioning.

And the violated expectation should be defined generically as:

> If there is a mismatch between the expected (calculated) value and the observed (measured) value at some output, it is a violated expectation.

The functional description should be useable to simulate the component behavior, i.e., to calculate the values of output ports if the values of the input ports are given. It should also be useable to infer the values of the input ports in terms of the values of other I/O ports. This is important if hypothetical reasoning is used for fault diagnosis. Though we have only used the functional description to calculate the value at the output port, our representation scheme can be used both ways.

The functional description is implemented as a LISP function, which calculates the desired port value in terms of the values of other ports. Every port of a component type has such a function associated with it, the link between the port and the function had been described in the structural description. Since different ports of different component types might have the same function, some functions can be shared. Several examples of the functional description as well as the SNePS rule which finds the violated expectation are as follows:

All annotations are shown in italics.

```
; Below is the function for the first output port of M3A2 type objects

(defun M3A2out1 (inp1 inp2 inp3)
    (plus (product inp1 inp2)
```

```
        (product inp1 inp3)))

; Below is for the single output port of MULTiplier type objects

    (defun MULTout1 (inp1 inp2)
        (product inp1 inp2))


; Below is an artificial example to show a function shared by several
; different component types such as the super-buffer, the wire or the
; 1-to-1 transformer.  All these component types show the same behavior
; at our level of component abstraction: echo the input to the output.

    (defun ECHO (inp1)
        inp1)



; The SNePS rule below is the only rule for concluding a violated
; expectation. it is actually part of the inference engine.  It is
; displayed here to show the benefit of the functional description of
; our current implementation.

    In SNePS code:

    (build
        avb ($p $vc $vm)
        &ant ((build port *p value *vc source calculated)
              (build port *p value *vm source measured))
        cq (build
              min 1 max 1
              arg (build name: THEY-MATCH p1 *vc p2 *vm)
              arg (build port *p state vio-expct]

    In English:

    If the calculated and measured values of port p are known as vc & vm,
    one and only one of the follows is true:
        1). vc and vm match;
        2). port p displays a violated expectation.
```

As depicted above, the functional description is versatile in that it supports the simulation and the inference of the device behavior; it supports hypothetical reasoning; and the representation scheme is quite simple.

3.2. Inference Engine

The inference engine for fault diagnosis follows a simple control structure. It starts from the top level of the structural hierarchy of the device, tries to find the output ports which show a violated expectation, and then uses the structural description to find a subset of components at next hierarchical level which might be responsible for

the bad outputs. The process is then mapped down to the suspicious parts, and a part is declared faulty if it shows some violated expectation at its output port and it is at the bottom level of the structural hierarchy, i.e. it is the smallest replaceable unit and there's no need to examine its details.

The inference engine is a rule-based system implemented in the SNePS Semantic Network Processing System. The control flow is enforced by a LISP driving function called "diagnose". SNePS can do both forward and backward inference, and it is capable of doing its own reasoning to diagnose the fault. The LISP driving function is introduced for execution efficiency.

A small set of SNePS rules is activated at every stage of the diagnosis. For example, three rules are activated when reasoning about a possible violated expectation of a specific port of a device. One rule is to deduce the measured value of the port. If the value can not be deduced from the wire connections, the rule would activate a LISP function which asks the user to supply one. A similar rule is for the calculated value, and the last rule is to compare the two values to decide if there is a violated expectation. The last rule has been shown in the section on functional description.

The diagnosis strategy along with the combination of a LISP driving function and SNePS rules turns out to be very effective. The diagnosis can be monitored by the SNePS text or graphic inference trace. The graphical trace is only available for Version 1, but will be implemented for Version 2. Another new feature of Version 2 is that it warns the user if the diagnosis is incomplete due to insufficient information.

3.3. Demonstration Example

An annotated demonstration is shown below. The target device is an M3A2 type board. The board has three input ports and two output ports, and it has five sub-parts: three multipliers and two adders. The multipliers and adders are DRU's, thus the device has only two levels in its structural hierarchy. The structure of the test device D1 is shown in Figure. 1.

; All annotations are shown in italics.

; Many output listings were removed and the SNePS inference trace
; was turned off so that the demonstration did not get too long.

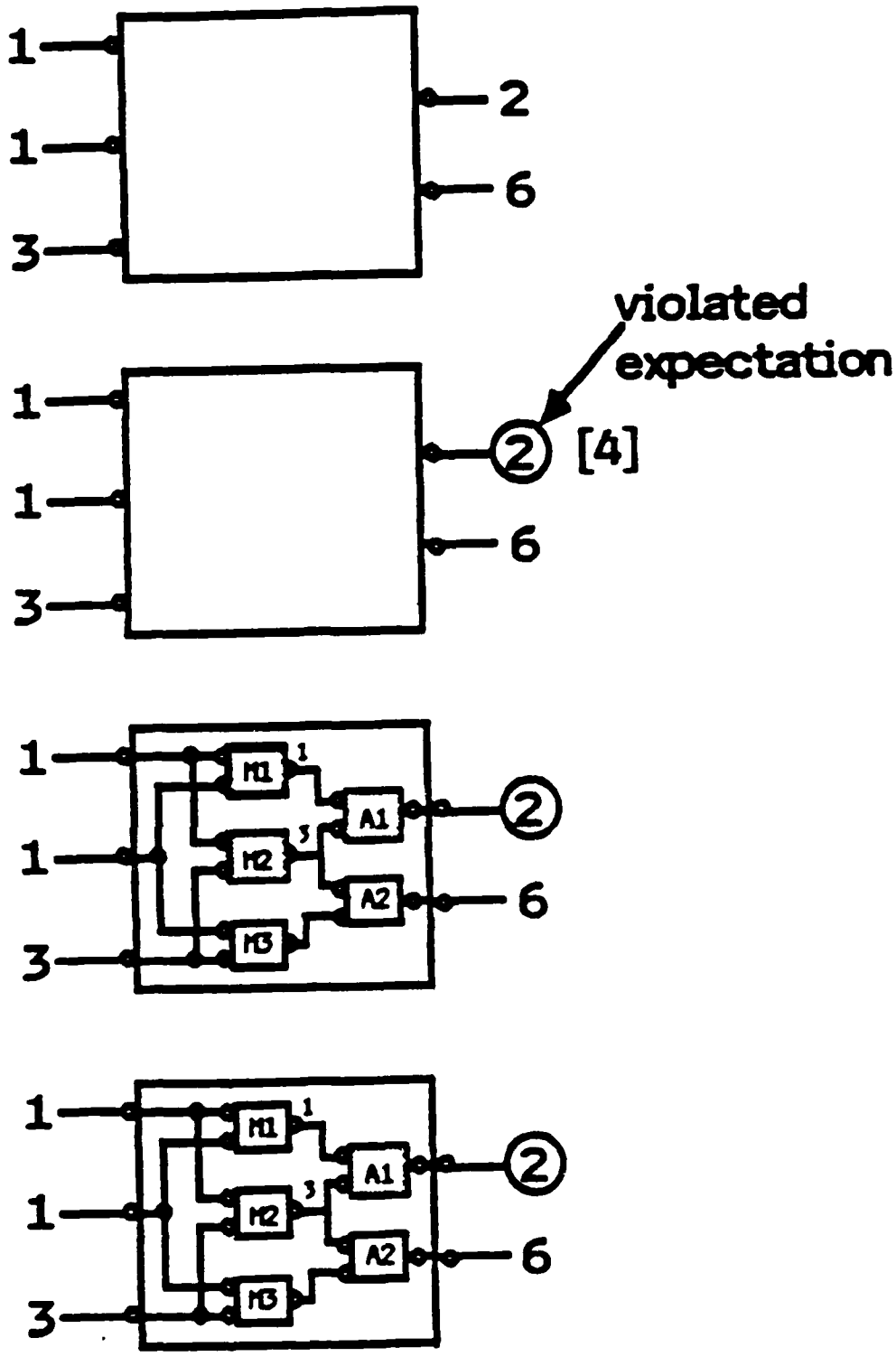
; Run the SNePS system which is written in Franz LISP.
; The computer used is a VAX 11/750 at Dept. of CS, SUNY at Buffalo.

% sneps
Franz Lisp, Opus 38.79
Thu Sep 12 20:37:18 1985

sneps

; The SNePS prompt is the asterisk.

; Bring in all arc definitions used by the network:
* (intext ARCS)
(done reading from ARCS)



A TEST PROBLEM

FIGURE 1

```
exec: 0.95 sec    gc: 0.00 sec

; Bring in the COMPonent library:
* (intext COMP)
(done reading from COMP)
exec: 19.08 sec    gc: 0.00 sec

; Bring in the CONTrol rules, i.e., the inference engine:
* (intext CONT)
(done reading from CONT)
exec: 7.03 sec    gc: 0.00 sec

; Load in the functional descriptions:
* (^(load 'FUNC))
t
exec: 1.21 sec    gc: 0.00 sec

; Declare the device D1 to be an M3A2 type object:
* (device-setup D1 M3A2)
; This activates the level-1 instantiate rule of the M3A2 type.
; and build the I/O ports and their function pointers as follows:
(m121 (state (TBI-L1)) (type (M3A2)) (object (D1)))
(m122 (inp-id (3)) (inport-of (D1)))))
(m123 (inp-id (2)) (inport-of (D1)))))
(m124 (inp-id (1)) (inport-of (D1)))))
(m126 (p3 (m122 (inp-id (3)) (inport-of (D1))))
    (p2 (m123 (inp-id (2)) (inport-of (D1))))
    (p1 (m124 (inp-id (1)) (inport-of (D1))))
    (pn (3))
    (f-rule (M3A2out2))
    (port (m125 (out-id (2)) (outport-of (D1)))))
(m128 (p3 (m122 (inp-id (3)) (inport-of (D1))))
    (p2 (m123 (inp-id (2)) (inport-of (D1))))
    (p1 (m124 (inp-id (1)) (inport-of (D1))))
    (pn (3))
    (f-rule (M3A2out1))
    (port (m127 (out-id (1)) (outport-of (D1)))))
done
exec: 32.06 sec    gc: 3.30 sec

; The follows builds the values of I/O ports of D1:
* (build port (find inport-of D1 inp-id 1)
        value 1 source measured]
* (build port (find inport-of D1 inp-id 2)
        value 1 source measured]
* (build port (find inport-of D1 inp-id 3)
        value 3 source measured]
; Next one should be 4, but a 2 is observed for this case:
* (build port (find outport-of D1 out-id 1)
        value 2 source measured]
* (build port (find outport-of D1 out-id 2)
        value 6 source measured]
```

; Begin the diagnosis session for device D1:
; The messages prefixed by @@@ are from the driving function.

* (diagnose D1)

@@@ diagnose D1: finding vio-expct .....
; If the SNePS inference trace were on, it would show that the system found
; the first output of D1 was a violated expectation, and the other was not.

@@@ adding TBI-L2 for D1 .....
; D1 is instantiated at level-2 since further investigation is needed.

@@@ adding TFS for D1 .....
; Now, a state called TFS (To Find Suspects) is added for D1. This
; activates the rules which find suspicious sub-parts of D1:

@@@ suspects created: (D1A1 D1M1 D1M2)
; Note that D1A2 and D1M3 are not suspects.

; The diagnosis process is mapped down to each suspect:

@@@ diagnose D1A1: finding vio-expct .....

What is the value of port
(m155 (inport-of (D1A1)) (inp-id (1)))
value/nil? 1

What is the value of port
(m156 (inport-of (D1A1)) (inp-id (2)))
value/nil? 3
; The system asks the user to supply measured values of ports if they
; can not be deduced from the wire connections:

@@@ adding TBI-L2 for D1A1 .....

@@@ D1A1 is faulty by vio-expct & DRU
; D1A1 is found to be faulty since it is a DRU and behaves abnormally.

@@@ diagnose D1M1: finding vio-expct .....

@@@ D1M1 shows no problem

@@@ diagnose D1M2: finding vio-expct .....

@@@ D1M2 shows no problem

; Note that both D1M1 and D1M2 are not instantiated at level-2
; since they do not show any violated expectation at their outputs.
; Also note that the values of their I/O ports are not requested since
; they can be deduced by the system.

```
; A final report is given by the system:
>>>>> I GOT THE FAULTY PARTS AS  >>>>>
(m237 (state (faulty)) (object (D1A1)))
(dumped)
exec: 398.35 sec    gc: 74.73 sec


; The user can check all intermediate and final results:
; only a small part of it is shown below.
* (desc *nodes - *oldnodes)
(m255 (arg
      (m254 (state (vio-expct))
            (port (m150 (out-id (1)) (outport-of (D1M2))))))
      (max (0))
      (min (0)))
; The first output of D1M2 shows no violated expectation.
(m253 (source (calculated))
      (value (3))
      (port (m150 (out-id (1)) (outport-of (D1M2)))))
; The calculated value of the first output of D1M2 is 3.
(m187 (to (m125 (out-id (2)) (outport-of (D1))))
      (from (m186 (out-id (1)) (outport-of (D1A2)))))
; A wire runs from 1st output of D1A2 to 2nd output of D1.
(m166 (sub-parts (D1M1) (D1M2) (D1M3) (D1A1) (D1A2)) (super-part (D1)))
; The sub-parts of D1 are D1M1, D1M2, D1M3, D1A1 and D1A2.
(m165 (state (TBI-L1)) (type (ADDER)) (object (D1A2)))
; D1A2 is an ADDER, and has been instantiated at level-1.
(dumped)
exec: 15.25 sec    gc: 0.00 sec

* (exit)
No files updated.
```

---

4. DISCUSSION

An important aspect of our research is to find a good knowledge-representation scheme to support diagnosis. Many researchers use standard predicate logic, but this has several drawbacks: the representation, the resolution technique, and the diagnosis assumptions seem fairly unnatural. We have implemented our system in the SNePS Semantic Network Processing System [16]. Advantages are: (1) structural and functional knowledge are integrated into a single network; (2) reasoning is done by rule-based inference; (3) diagnosis assumptions are handled in a natural way; (4) the deduction process can be monitored; (5) inference can also be traced graphically; (6) the representation can be easily expanded and modified; (7) procedural knowledge is represented and used; (8) it is smoothly interfaced with LISP.

The structural description is represented by instantiation rules at two different levels. This scheme turns out to be very effective and flexible. It can be used to pre-instantiate the target device with only little change. We ran the same example as the one used in the last section in both regular mode, which did the instantiation only when needed, and the pre-instantiation mode. As expected, the former was memory efficient, and the latter was good

for diagnosis speed. For the example of the M3A2 type device, the latter was four times faster than the former.

The main feature of our device representation scheme is the versatility of the system. To adapt the system to new devices, the only thing that needs to be done is to add new components to the system's libraries. In order to test this idea as well as the suitability of hierarchical structural representation, we invented a new device type called XM3A2 and put it into the system. The XM3A2 type has three inputs and two outputs, and only has a single sub-part which is of M3A2 type. Actually, it is a device which has an extra layer of packaging on an M3A2 type device. The M3A2 type has been known to the system, thus only the XM3A2 needed to be described to the system, and the description is two simple instantiation rules. There is no need for new functional description since the function of XM3A2 is the same as M3A2. The device has three levels of structural hierarchy, and our test successfully found the faulty part at the lowest level. Though the example of XM3A2 is somewhat simple, it displays the capability of the system to deal with a wide range of devices in the domain with arbitrary complexity.

5. FUTURE WORK

A potential problem is that this approach to fault diagnosis is only good for digital circuitry without feedback. There are many devices that are mixtures of digital and analog circuitries. To adapt the system to those devices may require some modification of the device representation scheme. The representation and use of second principle rules should also be introduced for better system performance.

In our current scheme, similar component types, which have the same function but different specifications, are represented individually. An example is the representation of 1-to-1, 1-to-2, and 1-to-3 transformers. It would be better to represent all types of transformers by a single representation with a parameter to specify the transforming rate.

There is no user interface for adding new components so far. The development of a formal language for device representation may solve this problem as well as others. The language should support all diagnosis related activities, such as device simulation and structure retrieving. It should also support the system construction and adaptation.

Part. II Graphical Interface of VMES

1. INTRODUCTION

The main mode of communication between the SNePS reasoning mechanism used for the VMES project and the user is intended to be a graphical interface. We have implemented a new version of such an interface called "SENDING". This version supersedes the version of SENDING that has been described in our final report of the Post Doctoral project (SCEEE) [18].

The first change noticeable between these two versions is the changed domain. In our effort to construct a *versatile* program we have changed the "device" of diagnosis. While the SCEEE world consisted of a small number of logic gates which were only partially connected, we have since then tackled two new devices.

The first one of these devices is a little Adder/Multiplier that has been a test object of several researchers in the field of trouble shooting. The second one is a piece of a real device, a 6 channel PCM board. This change of domain however was not the crucial step.

Only insignificant improvements have been made to the representation of visual knowledge. The basic case frames are still the same as described in the SCEEE report, and so are the used relations. The significant changes in the second generation of SENDING are an improvement in speed by approximately a factor of seven of the display program, and a considerable expansion of the power of both, the display and the readform function. This section of our report will first make some general comments on "visual knowledge" and will then continue by describing the new options of "display".

While visual knowledge has been dealt with implicitly in computer vision and from a different aspect in computer graphics, and in cognitive psychology for quite some time, we lately have been experiencing a growing interest in an explicit treatment based on Knowledge Representation methods [2, 9]. The crucial point here is the interest in a natural representation that lends itself to reasoning processes as opposed to a representation for ease of "recognition" or of "display". Some more references on this subject are given in the special section on "Other Activities", at the end of this paper, dealing with the acquisition of background knowledge.

2. THE GRAPHICAL INTERFACE

2.1. Motivation

Why should somebody want to implement a program like "SENDING" (SEmantic Network *Domain Interface Graphics*)? Our interest in this interface is twofold. Currently there is a growing interest in multi media communication [14]. Technical literature would be impossible without charts, diagrams and drawings. It seems that also a dialog between a technician and an advisory expert system about a technical object like a circuit board would profit very much from a graphical component.

One can even go so far to say that diagrams are the "interlingua" of the technical literature. The display of the device under repair can be used in our system by both the user and the computer to refer to parts which are currently under discussion.

The second source of our interest in graphical interfaces is of theoretical nature. We are investigating principles of visual knowledge representation. In computer vision or computer graphics, representations are mainly designed in order to permit efficient

recognition or display of objects. We are interested in representations that can be used in *reasoning*, about forms as well as for display purposes.

2.2. Components of the Interface

The SENDING graphical interface contains several parts, the most important of which are the "display" function and the "readform" function. The readform function is our (simple) version of a CAD device. It permits a user to create a simple object, consisting of arcs, lines, circles, boxes, text, etc. by drawing them on the screen of a graphics terminal. Objects can contain several unconnected parts and are stored immediately as named objects, namely as LISP functions.

Although it is not our purpose to compete with any of the very fancy existing CAD systems, considerable improvements to readform have been made over the last year. Currently work is done on the third generation of readform. Only the introduction of arcs made it possible to design most of the common logic symbols (AND and OR gates) and of transformers. In earlier described versions of SENDING a separate libarary was necessary for round objects.

Improvements currently worked on are commands that make the creation of repetitive structures easier. Also earlier defined objects can be loaded into currently built up more complex objects.

The logical counter part of readform is the "display" function. Display takes one or more nodes of a semantic network as arguments. These nodes can be either base nodes, representing objects, or assertion nodes, representing simple propositions about one object. Assuming the semantic network contains propositions about form, position and attributes of an object, "display" can retrieve this information and create a picture of the object on the screen. Displayable propositions also have to say something about form/position of an object, and the display of the proposition is done by showing the described object.

It should be noted that this approach to image generation is different from the techniques usually employed by computer graphics programs. Our object descriptions are given in a declarative format, incorporating them together with a part and a type hierarchy into a single network. We are comparing this approach to graphics with language generation from an internal knowledge representation. Such a language generation program takes a semantic network as its input and generates a surface utterance from it. The difference here is, that a picture is generated.

2.3. How display works

The "form" itself is a LISP function (created by readform), which is represented in the semantic network as a base node whose node label is identical to the function name. (For explanations of the SNePS terminology refer to the given reference about SNePS [16]).

The detailed process of displaying an object is: first the part hierarchy is used to retrieve subparts of the given object; then forms and positions of all parts are retrieved. We are permitting several different methods of positioning which are expressed with different case frames in the network. The simplest case is absolute positioning in device coordinates. More involved are relative position of an object to another object or to its super-object. The most complicated version retrieves the relative position of a part relative to its super-part by using the type hierarchy that part and super-part belong to.

After knowing position and form, attributes of objects are retrieved. Attributes can be either symbolic attributes or iconic attributes. An iconic attribute is directly displayable, and the simplest form of such an attribute is "color". Symbolic attributes have to be mapped into iconic attributes, in order to make them displayable. For instance we are marking faulty objects by changing their genuine color into a signal color (red). In this case the same medium (color) is used to express a different fact.

Attributes in our system are treated in a way that we have not seen described in the literature before, namely by making the attribute class itself a LISP function. An attribute value is passed to this LISP function as an argument (sometimes a dummy value), together with the form function, effectively making the attribute-class function a functional. The returned value of the attribute-class function is again a form function, but it is modified according to the given attribute.

Our approach to attributes guarantees that we can apply new predicates to old forms, without ever changing the form-functions. Any alternative that comes to mind would require adding new parameters to form-functions. More details on the case frames used for form position attributes can be found in the repeatedly quoted SCEEE report.

2.4. Special display Parameters

2.4.1. Modality

The display function permits the user to specify a number of different parameters. One is a "modality" parameter. In our maintenance domain we are dealing with structural and functional properties of objects. This implies that it is possible and desirable to display objects in both these aspects (or as we say, modalities). The user can select which of the stored aspects he wants to see, by specifying the modality parameter accordingly. Functional display is the default.

The modality parameter is perfectly general and can be extended to any number of different aspects, however we currently see no need for others than structural and functional displays. Assertions for different modalities are not structured in a Hendrix type [6] partition system but they contain a modality slot in the object description case frame.

Our current research has led us to the result that structural and functional displays should be treated differently, and we will talk about this more in the section on future work.

2.4.2. Pruning the display

If a display function is used as an intelligent system as opposed to a simple mapping from a data structure to a display device, there has to be a way to "prune" the display to avoid "overloading" the user, by presenting irrelevant and therefore confusing information. (One of our goals in this project is to find a method to create a cognitively appealing representation that limits the displayed information to relevant objects and relations).

Several optional parameters for display have been defined, that permit the user to control the amount of information that he receives. Our goal is to automatize this process entirely, but currently the user has to decide himself what he considers appealing. The following paragraphs contain a description of these user options.

As mentioned before, our representation uses a part hierarchy. A "level" parameter permits the user to limit the number of levels in the part hierarchy that are displayed. If, for example, an object has sub-parts which have sub-parts in turn, it is possible to limit the display to showing only sub-parts, but not their sub-parts (i.e. the sub-sub-parts of the object are not shown). Any number of levels can be represented in the semantic network, and correspondingly any natural number can be specified for the level parameter. Figure 1 displays our Adder/Multiplier board at :level 2. Figure 2 shows the same Adder/Multiplier at :level 3.

Sometimes the number of effectively *visible* objects might be responsible for overloading of the user. Therefore an "objects" parameter limits the number of (sub)objects displayed. As in the level case, objects are retrieved from the part hierarchy by using breadth first selection. If the specified number of objects has been shown, display will terminate in the midst of a level.

In our current representation there is no way to express different importance for different sub-parts; therefore an "object" parameter results sometimes in displaying "unimportant" parts, a problem which has been criticized by several users. We plan to investigate this question in the future.

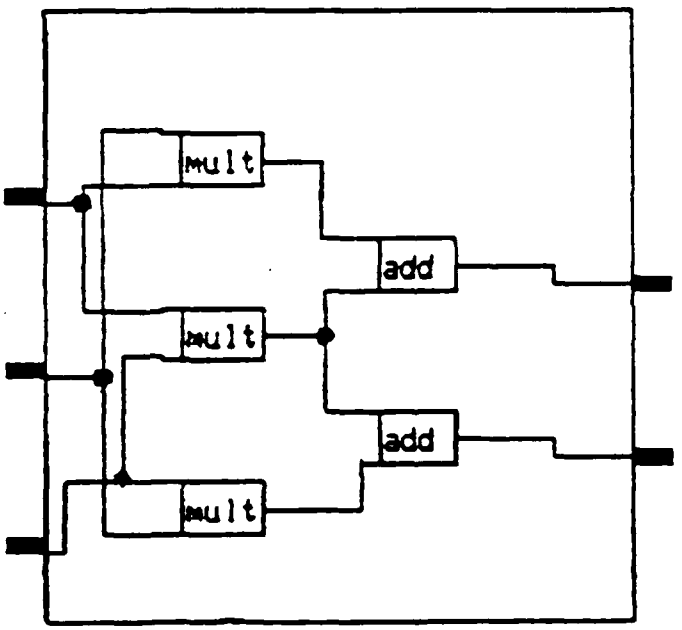


Figure 1: Adder/Multiplier at level 2

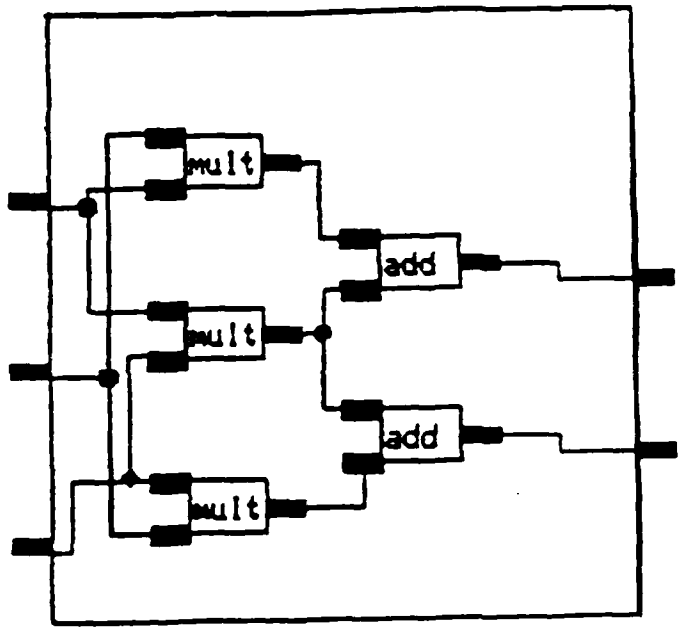


Figure 2: Adder/Multiplier at level 3

Objects in the VMES system can themselves be of quite varying complexity. A simple wire is an object, but a 16 leg integrated circuit is also one object. In order to take care of this problem another display option has been programmed, the "complexity" parameter.

Display's "complexity" parameter extends the ideas developed above by counting not the number of objects, but the number of graphical primitives contained in them. So it is possible for the user to limit the number of *graphics primitives* that are displayed. In this way two display calls with the same "complexity" parameter might create either a picture of a simple object with five sub-parts, or a picture of a complicated object with only one sub-part.

2.4.3. Optimal screen use

Another type of display option deals with the use of the given screen space, the so called "fill" option. If display is called with the "fill" option, it dynamically computes its own window to viewport mapping to guarantee an optimal use of the given (globally specified) viewport. This option is also the only way to display parts of the world that do not fit into screen coordinates. In this way a user sees small objects at a reasonable size, while large objects still fit into the screen. Still he does not have to know anything about viewports and windows.

The "fill" option also permits us to avoid another common problem in computer graphics. If a window is defined arbitrarily, chances are that some of the displayed

objects will be cut into two parts, only one of which is inside the window. This requires in commonly used graphics packages the very time consuming activity of "clipping". We also think that the average user is not interested in half objects. All the things that he specifies because he wants to see them, he wants to see in whole. All the objects he does not specify he either does not want to see at all, or at least he does not mind if they are not shown to him (in half!). Cutting objects into parts disagrees with our whole object oriented approach to AI.

The way all this is achieved is by having display/fill compute an optimal window in the world which completely surrounds all desired objects with the smallest possible rectangular extent. This window is mapped into the supplied viewport, using the same scale factor for both x and y coordinates. This guarantees filling the viewport in one of these two dimensions. (Note that in order to fill it in both dimensions distortions would be necessary, which might show a circle as an ellipsis. This is not only optically undesirable, but also difficult to compute).

An extension of the "fill" option is the "intell" option. It constitutes another step in giving the system possibility to decide what to display. Although the name "intell" seems a little bit pretentious, it is definitely a step towards having the system figure out what the user really wants to see as opposed as to what he is asking for. The intell option is the solution for the following problem. If a user requests to see a certain object, he might at the same time be interested to see where this object "fits into the whole".

A user might also want to know if there are several other objects of the same type. If display is called with the "intell" option it will display the user specified object(s) in one viewport and in another viewport, will show the chain of all super-objects of the user specified object(s). Currently the default viewports are the left half of the screen for the object, and the right half of the screen for the super-objects. Every super-object will be shown to two levels depth (see "levels" above). So if a user displays a leg of an AND gate, then the AND gate with all its ports "(legs)" will be displayed. If the super-object of the AND gate is a board, then the board will be displayed with all its gates, but not with their legs. The use of the "intell" option is shown in Figure 3. Figure 3 and all other figures were created with a printer that directly dumps a screenful from a graphics terminal. It shows a multiplier displayed in the left viewport, and the corresponding Adder Multiplier board in the right viewport. Finally Figure 4 shows the 6 channel PCM board in the right viewport and one of its PCM chips in the left viewport.

2.5. Graphical Inference Trace

The SNePS system has a tracing facility which permits a user to watch the reasoning process of SNePS. The function that is used for tracing is independent of SNePS, and it is possible to plug different interfaces into this position. An important aspect of display is that it can be used as such an interface. In other words, an observer can watch what SNePS is currently "thinking" about.

In our implementation of a diagnosis system for the Adder Multiplier board that we have mentioned above, the system marks parts that it is currently "thinking" about by displaying a question mark above them, and parts that it found a conclusion about by showing an exclamation mark above them. The faulty part is shown in the final display in red.

This is a direct consequence of SNePS figuring out that the part is bad. Using the attribute mechanism described above, the "state" attribute class is automatically

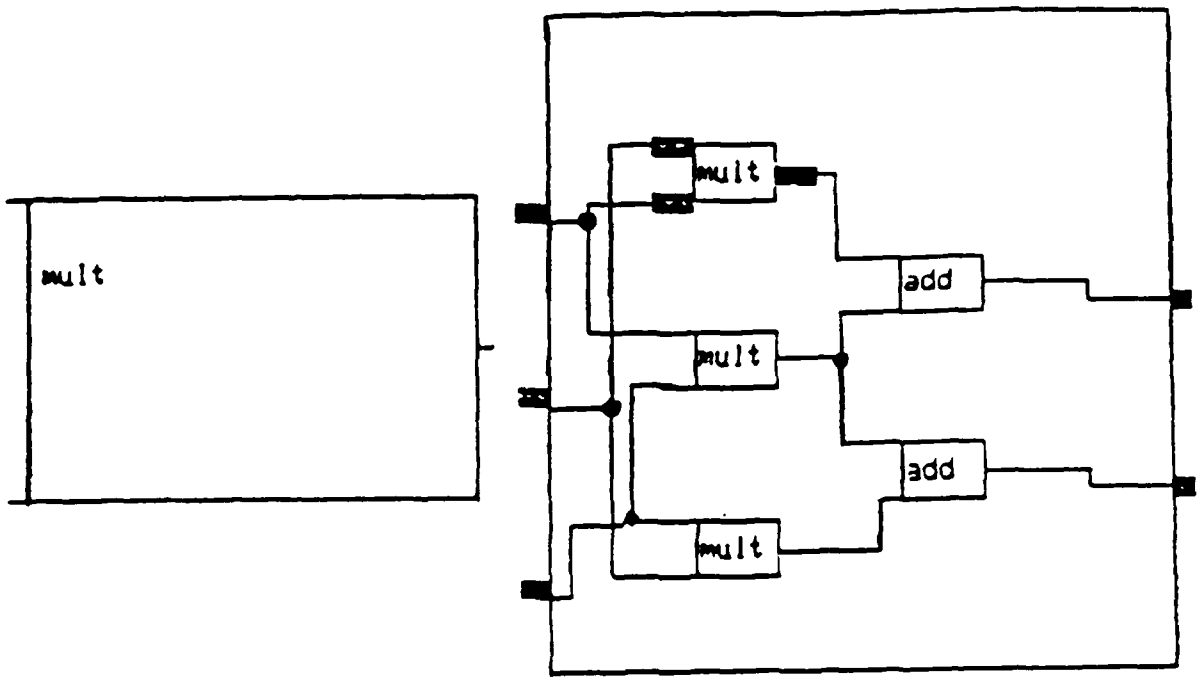


Figure 3: A Multiplier and its Board

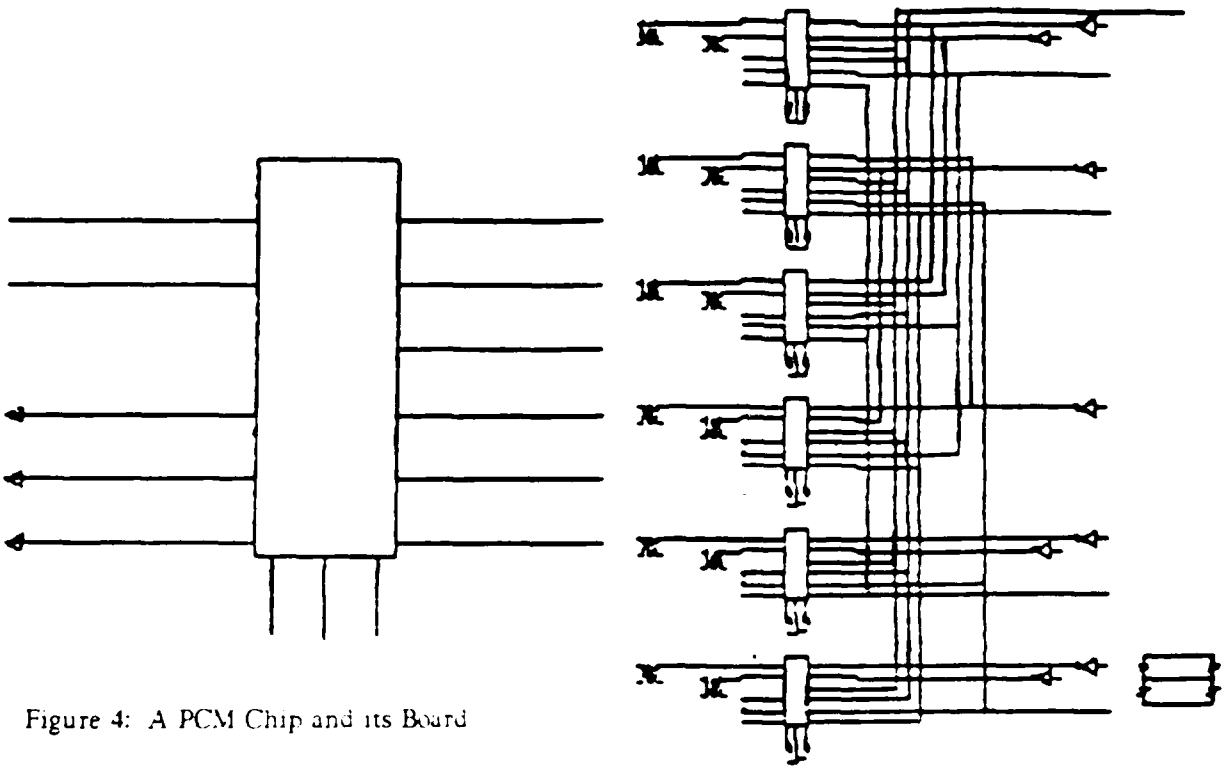


Figure 4: A PCM Chip and its Board

translated into the signal color red. After the reasoning process has terminated, any display command of the object found faulty will again be in the new color. This is the case, because the semantic network has been changed permanently by the reasoning process. The mechanism of infertrace does of now not work for the PCM board for which we use a much more complicated representation system which has created unexpected interactions.

3. FUTURE WORK

Our future plans include the investigation of the knowledge representation scheme for display purposes. We also have noted interesting differences between structural and functional displays. These differences have to do with the different types of knowledge that have to be specified. While structural displays require considerably more fixed coordinate values, functional displays can replace this type of knowledge by knowledge about object clusters and their inner workings.

References

1. Ronald J. Brachman and James G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, ().
2. Randall Davis, Howard Shrobe, and al., "The Hardware Troubleshooting Group," *SIGART Newsletter* **93**(Jul. 1985).
3. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence* **24** pp. 347-410 (1984).
4. M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence* **24** pp. 411-436 (1984).
5. R. T. Hartley, "CRIB: Computer Fault-finding Through Knowledge Engineering," *Computer*, (March 1984).
6. Gary G. Hendrix, "Encoding Knowledge in Partitioned Networks," pp. 51-92 in *Associative Networks: Representation and Use of Knowledge by Computers*, ed. Nicholas Findler, (1979).
7. E. C. Kingsley, N. A. Schofield, and K. Case, "SAMMIE - A Computer Aid for Man Machine Modeling," *Computer Graphics* **15**(3)(Aug. 1981).
8. Stephen M. Kosslyn and Steven P. Shwartz, "A Simulation of Visual Imagery," *Cognitive Science* **1** p. 265 (1977).
9. Andrew Latto, David Mumford, and Jayant Shah, *The Representation of Shape*, IEEE Workshop on Computer Vision Representation and Control (1984).
10. Hector J. Levesque, "Foundations of a Functional Approach to Knowledge Representation," *AI* **23**(1984).
11. Alan K. Mackworth, "On Reading Sketch Maps," *IJCAI - 77*, (1977).
12. Robert Milne, "Fault Diagnosis through Responsibility," *IJCAI*, pp. 423-425 (1985).
13. Fanya S. Montalvo, "Visual Knowledge Representation and Acquisition," *Second Annual Meeting of the Cognitive Science Conference*, p. Session #12 (1980).
14. Joyce K. Reynolds, Jonathan B. Postel, Alan R. Katz, Gregg G. Finn, and Annette L. DeSchon, "The DARPA experimental multi media mail system," *Computer*, pp. 82-91 (Oct. 1985).
15. Ethan A. Scarl, John R. Jamieson, and Carl I. Delaune, "Process Monitoring and Fault Location at the Kennedy Space Center," *SIGART Newsletter* **93**(Jul. 1985).
16. S. C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and Use of Knowledge by Computers*, ed. Nicholas V. Findler,Academic Press, New York (1979).
17. S. C. Shapiro and The SNePS Implementation group, *SNePS User's Manual*, Department of Computer Science, SUNYAB, Buffalo, NY (revised: 1983).
18. Stuart C. Shapiro, Sargur N. Srihari, James Geller, Ming-Ruey Taie, Chi Choy, and Albert Hanyong Yuhan, "A Graphics Interface to a rule-based system," SCEEE-PDP 84-30 SCEEE-PDP/84-31, Southeastern Center for Electrical Engineering Education, St. Cloud, FL 32769 (Jan. 1985).
19. E. H. Shortliffe, G. Adorni, A. Boccalatte, and M. Di Manzo, "Cognitive Models for Computer Vision," *COLING - 82*, American Elsevier North Holland, (1982).
20. Hideyuki Tamura, "Image Database Management for Pattern Information Processing Studies," in *Pictorial Information Systems*, ed. K.S. Fu, ().

A Fault Diagnosis System Based on an Integrated Knowledge Base[1]

Stuart C. Shapiro, Sargur N. Srihari, James Geller, Ming-Ruey Taie

Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
CSNET: taiemr%buffalo@csnet-relay

Goal of the Project. We are developing a versatile maintenance expert system (VMES) for trouble-shooting electrical circuits. VMES is able to diagnose a variety of common faults of electrical equipment. It can easily be adapted to new devices, and can communicate with the user through natural language, graphics and menus.

Structural and functional descriptions, usually referred to as "design models" of a device, build one type of knowledge in our integrated knowledge base. This kind of knowledge has been widely used by fault diagnosis researchers as a solution to the difficulties of empirical-rule-based diagnosis systems in knowledge acquisition, diagnosis capability, and system generalization [1]. The knowledge needed for building such a system is readily available and well-structured. Empirical associations of individual devices are excluded due to their highly device-specific nature, which impairs versatility.

Domain knowledge has long been ignored by researchers of design model based systems. This sort of knowledge is not part of the design model, yet it consists of common rules for the domain. Though it may still be empirical, it can be applied to any device in the domain. An example is that a burnt appearance of a low level component, such as a resistor, will usually imply that it is a faulty part.

System Architecture. An important aspect of our research is to find a good knowledge representation scheme for fault diagnosis. Many researchers use predicate logic, but this has several drawbacks: the representation, the resolution rule, and the diagnosis assumptions seem fairly unnatural [2]. We are implementing our system using SNePS the "Semantic Network Processing System" [3]. Advantages are: (1) structural and functional knowledge is integrated into a single network; (2) the powerful SNePS non-standard connectives permit us to express rules of a degree of complexity which most other Horn clause based systems cannot use; (3) diagnosis assumptions are handled in a natural way; (4) the deduction process can be monitored; (5) inference can be traced graphically; (6) the representation can be easily expanded and modified; (7) procedural knowledge is represented and used.

VMES consists of an integrated knowledge base and a device independent inference engine. A hierarchically arranged knowledge base provides abstraction levels of devices, and makes the inference engine able to focus on a limited number of objects at any time. Initially only component *types* are represented in the knowledge base, an object is instantiated only when needed. Since devices in the domain share common components, this approach avoids redundant representations. When the system is adapted to a new device, the only thing needed is to add descriptions of the new component *types* used by the new device.

An important part of VMES is its graphical interface which comprises a separate subsystem called as "SENDING" (SEmantic Network Domain Interface Graphics). The main part of this interface is a function "display" which generates a picture from a semantic network representation. Unlike the usual computer graphics approach to image generation, it is more comparable to a language generation program that takes a semantic network as its input and

[1] Work reported here was supported in part by the Air Force Systems Command, Rome Air Development Center and the Air Force Office of Scientific Research.

generates a surface utterance, in this case a picture, from it.

The display function permits the user to specify a number of different parameters. One is a "modality" parameter which permits the user to select whether the structural or functional aspect of a device should be displayed.

If a display function is used as an intelligent system as opposed to a simple mapping from a data structure to a display device, there has to be a way to "prune" the display to avoid "overloading" the user, by presenting irrelevant and therefore confusing information. One of our goals in this project is to find a method to create a cognitively appealing representation.

Our representation uses a part hierarchy. A "level" parameter permits the user to limit the number of levels in the part hierarchy that are displayed. If a part has sub-parts which have sub-parts in turn, it is possible to limit the display to showing only sub-parts, but not their sub-parts. Sometimes the number of effectively *visible* objects might be responsible for overloading of the user. Therefore an "objects" parameter limits the number of (sub)objects displayed using breadth first selection.

Display's "complexity" parameter limits the number of *graphics primitives* that are displayed. In this way two display calls with the same "complexity" parameter might create either a picture of simple object with five sub-objects, or of a complicated object with no sub-objects. If display is called with the "fill" option, it dynamically computes its own window to viewport mapping to guarantee an optimal use of the given (globally specified) viewport.

If display is called with the "intell" option it will display the user specified object(s) in one viewport and in another viewport, will show the chain of all super-objects of the user specified object(s) to a depth of two levels. This permits the user to find out where the object he wanted to see is, relative to the "whole" object he is dealing with. An important aspect of display is, that it can be used by SNePS, to show SNePS' reasoning graphically. In other words, an observer can watch what SNePS is currently "thinking" about.

A version of VMES has been developed and used successfully to pinpoint a faulty adder of an adder/multiplier board, which is a favorite artificial example of researchers in this domain. In this version the system marks parts that it is currently thinking about by displaying a "?" above them, and parts that it found a conclusion about by showing an "!" above them. The faulty part is shown in the final display in red.

Future Plans. We successfully adapted the system to a simplified real device, a six channel PCM board used for telephone communication. Our future plans include the investigation of domain knowledge, and further development of the knowledge representation scheme for reasoning and display.

References

1. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence* **24** pp. 347-410 (1984).

2. R. Fikes and T. Kehler, "The Role of Frame-Based Representation in Reasoning," *Communications of the ACM* **28**(9) pp. 904-920 (Sep. 1985).

3. S. C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and Use of Knowledge by Computers*, ed. Nicholas V. Findler,Academic Press, New York (1979).

A Logic for Belief Revision

by

João P. Martins [*] and Stuart C. Shapiro [**]

[*] Departamento de Engenharia Mecanica
Instituto Superior Tecnico
Av. Rovisco Pais
1000 Lisboa, Portugal

[**] Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260, U.S.A.

Abstract

Artificial Intelligence (AI) is the branch of Computer Science which studies how to enable computers to exhibit "intelligent" behavior. One of the central problems in AI research, *automated reasoning*, is that of how computers can automatically draw conclusions from bodies of information. The methodology for addressing this problem requires choosing a general area of application, choosing a representational scheme for the propositions in the domain of application, and developing methods for deriving new information from old information. One branch of automated reasoning is concerned with what to do when a contradiction is found during the reasoning process.

Systems that are able to detect contradictions and/or identify their causes, called Belief Revision Systems, Truth Maintenance Systems, or Reason Maintenance Systems, have been implemented by several researchers in AI. Most of the AI work done in the field of belief revision is characterized by being directed towards the implementation of belief revision systems, without explicit concern about the theoretical foundations of such systems. In this paper, we present a logic suitable to support belief revision systems and discuss the properties of a belief revision system based on that logic.

One of the fundamental problems that any logic underlying a belief revision system has to address is how to keep track of and propagate propositional dependencies. This is important, because, in the event of detection of a contradiction, one should be able to identify *exactly which* assumptions were used in the derivation of the contradictory propositions, in order not to blame some assumption irrelevant to the occurrence of the contradiction as the culprit for the contradiction. In the field of logic, the relevance logicians have also been interested in keeping track of what assumptions were used to derive any given proposition; they have developed mechanisms for this, as well as for preventing the introduction of irrelevancies. The

SWM system, described in this paper, is based on relevance logic and makes use of the book-keeping mechanisms of relevance logic. In addition, SWM has some other mechanisms to remember the contradictions that were derived, to prevent their re-derivation.

The SWM formalism guarantees that (1) Each wff is associated with every hypotheses that was used in its derivation. (2) Each wff is only associated with the hypotheses that were used in its derivation. (3) Each wff is associated with every set known to be inconsistent with the hypotheses used in the derivation of the wff. (4) The application of the rules of inference is blocked if the resulting wff would be dependent on a set of hypotheses known to be inconsistent.

1. Background

The ability to reason about and adapt to a changing environment is an important aspect of intelligent behavior. Most computer programs constructed by researchers in AI maintain a model of their environment (the external and/or internal environments), which is updated to reflect the perceived changes in the environment. The model of the environment is typically stored in a knowledge base (containing propositions about the state of the environment), and the program manipulates the information in this knowledge base. Most of the manipulation consists of drawing inferences from information in the knowledge base. All the inferences drawn are added to the knowledge base. One reason for model updating (and thus knowledge base updating) is the detection of contradictory information about the environment. In this case, the updating should be preceded by a decision about which proposition in the knowledge base is the culprit for the contradiction, its removal[1] from the knowledge base and the subsequent removal from the knowledge base of every proposition that depends on the selected culprit.

Systems that are able to detect contradictions and/or identify their causes, called Belief Revision Systems, Truth Maintenance Systems, or Reason Maintenance Systems, have been implemented by several researchers in AI. Most of the AI work done in the field of belief revision is characterized by being directed towards the implementation of belief revision systems, without explicit concern about the theoretical foundations of such systems. In this paper, we present a logic suitable to support belief revision systems and discuss the properties of a belief revision system based on that logic.

Specifically, in this paper we discuss the following issues: What kind of logic should underlie a computer program used for applications in belief revision? What kind of provisos should be supplied by the logic in order to cope with the possible occurrence of contradictions?

[1] Or making inaccessible to the program.

How should a computer program based on that logic interpret those provisos in order to recover from contradictions and to avoid performing reasoning from contradictory hypotheses? How should a computer program based on that logic distinguish between propositions believed by different individuals?

Section 1 of the paper describes the background work in AI related to belief revision and discusses problems existing in previously developed systems; Sections 2 through 11 present a logical system, the *SWM system*, that relies on the notion of dependency and provides for dealing with contradictions. The SWM system is loosely based on relevance logic and associates each proposition with the set of hypotheses that were *really* used in its derivation and with the sets of hypotheses with which it is incompatible. The rules of inference of SWM are stated so that they prevent the combination of propositions known to be incompatible. Sections 11 through 13 discuss the features that a computer system based on SWM will exhibit. These features define the behavior of an abstract belief revision system, which we call MBR (**M**ultiple **B**elief **R**easoner). MBR is an abstract system in the sense that it is described independently of its implementation. In MBR a *context* is any set of hypotheses. A context determines a *belief space* , which is the set of all propositions depending exclusively on the hypotheses defining the context. We discuss how the contradiction-handling features of SWM are reflected in the notions of context and belief space. We also discuss how such notions can be used to represent the beliefs of different individuals in a common knowledge base, maintaining a clear distinction between them. Finally, Section 14 presents an example obtained using SNeBR, a particular implementation of MBR.

The conventional approach to handling *contradictions* consists of changing the most recent decision made, i.e., the contradiction is blamed on the most recent decision made (chronological backtracking). An alternative solution (dependency-directed backtracking) consists of changing, not the last choice made, but the choice that *most likely* caused the unexpected condition to occur. This second approach, proposed in the late 70's by Stallman and Sussman ori-

ginated a great deal of research in an area of AI that has become known as "belief revision".

Belief revision is an area of AI research concerned with the issues of revising sets of beliefs when new information is found to contradict old information. Research topics in belief revision include the study of representation of beliefs, in particular how to represent the notion of belief dependency; the development of methods for selecting the subset of beliefs responsible for contradictions; and the development of techniques to remove some subset of beliefs from the original set of beliefs. *Belief revision systems* [Doyle and London 80; Martins forthcoming] are AI programs that deal with contradictions. They perform reasoning from the propositions in a knowledge base, "filtering" these propositions, so that only part of the knowledge base is "perceived", namely, the set of propositions which are under consideration. This set of propositions is usually called the set of *believed propositions*. When the belief revision system decides to consider another of these sets it is usual to say that it *changes its beliefs*. Typically the belief revision system explores alternatives, makes choices, explores the consequences of its choices, and compares results obtained when using different choices. If a contradiction is detected during this process, then the belief revision system will revise the knowledge base, "erasing" some propositions to get rid of the contradiction.[2]

Belief revision systems in AI have their roots in the frame problem [McCarthy and Hayes 69, Raphael 71, Hayes 73], the problem of deciding which conditions change and which conditions do not change when a system undergoes some modification. The basis of the problem is that although it is possible to specify the ways in which a system's environment might change in terms of the effects of actions, some way of deciding what remains unchanged by the actions must also be specified. For example, when planning a sequence of actions to stack a red block on a green block, it is obvious that one needs an axiom that asserts that a block is where you put it after you move it there. It is less obvious that one also needs an axiom that

[2]There are some cases in which it is desirable to continue reasoning within a knowledge base in which a contradiction exists, for a description of this type of reasoning refer to [Martins 83].

asserts that the color of a block doesn't change after it has been moved. And what about the other myriad facts about the world that don't change when you move a block?

A significant milestone towards the development of belief revision systems is the work of Stallman and Sussman, who designed a system, called EL, in which dependencies of propositions are permanently recorded [Stallman and Sussman 77]. EL maintains a complete record (trace) of its reasoning, using it both to decide which alternative choices to make when something goes wrong and to explain its line of reasoning. Along with each derived proposition, EL stores the set of all propositions directly used in its derivation and the rule of inference used to derive it, the *dependency record* of the proposition. EL solves electric circuit problems. While searching for the values of the circuit parameters, EL may have to "guess" the operating range of some devices. Later on, if an inconsistency is found, EL knows that somewhere along the way it guessed a wrong state for some device. The novelty of EL's approach to backtracking is that the assumption that is changed during backtracking does not necessarily correspond to the last choice made but rather to the assumption that *most likely* caused the contradiction to occur (*dependency-directed backtracking*). When an inconsistency is detected, EL searches through the chain of dependency records of the inconsistent propositions until it finds all the assumptions (guesses made) upon which the inconsistent propositions depend. Heuristics are then used to rule out one of them. This set of assumptions is recorded as leading to a contradiction and is never tried again.

Stallman and Sussman's work had two major influences in AI: (1) It opened a new perspective on the handling of alternatives (dependency-directed backtracking). (2) It influenced the creation of systems that handle and can recover from contradictions (belief revision systems).

Building upon Stallman and Sussman's work, Doyle designed the *Truth-Maintenance*

System (TMS) [Doyle 78, 79, 80], the first domain-independent belief revision system.[3] TMS maintains a knowledge base of propositions, each of which is explicitly *marked* as believed or disbelieved. TMS may be told that some propositions are contradictory, in which case it automatically revises its beliefs so that no inconsistent set of propositions is simultaneously believed.

TMS is based on the definition of two kinds of objects: *propositions* and *justifications*. Justifications represent the reasons why TMS believes or disbelieves a certain proposition. Attached to each proposition in the knowledge base, there is one (or more) justification(s) that supports TMS's belief or disbelief in that proposition. A justification contains two lists of propositions, the inlist and the outlist.[4] A proposition is *believed* if and only if every proposition specified in the inlist is believed and every proposition specified in the outlist is disbelieved. Whenever a proposition is derived, a justification is added to the proposition, containing all the propositions ***directly*** used in its derivation and the rule of inference used to derive it.

The two main actions that TMS may be called upon to perform are the addition of a new proposition to the knowledge base or the addition or retraction of a justification to a proposition. In either case, TMS tries to find (1) disbelieved propositions that will become believed by such addition or retraction and (2) believed propositions that will become disbelieved by such addition or retraction.

TMS may be told that a proposition and its negation are both believed. In this case, the dependency-directed backtracking mechanism of Stallman and Sussman is invoked: it searches through the knowledge base, starting with the justifications of the contradictory propositions, until it finds all the assumptions that are used by the contradictory propositions. One of those assumptions is selected as the culprit for the contradiction and is ***disbelieved***. To disbelieve this assumption, TMS believes in one of the propositions referenced in the outlist of the

[3]The field of belief revision in AI is usually recognized to have been initiated by the work of Doyle, although a system that performs belief revision (in robot planning) was developed simultaneously by Philip London [London 78].

[4][Doyle 79] discusses other types of justifications which were not implemented in TMS.

assumption and justifies this proposition with a justification whose inlist contains the proposition representing the contradiction. After selecting the culprit for the contradiction, it is necessary to disbelieve all the propositions depending upon it. This is done by following the chain of dependency records and disbelieving each proposition that has no justification other than the one that includes the selected culprit in its inlist. This "disbelieving process" is not as simple as it may seem, due to the possibility of circular proofs. See, for example [Charniak, Riesbeck and McDermott 80, pp.193-226].

Doyle's research triggered the development of several belief revision systems [Goodwin 82, 84; McAllester 80; McDermott 82; Shrobe 79; Thompson 79]. These systems share two characteristics: (1) They are mainly concerned with implementation issues, paying no special attention to the logic underlying the system. (2) Each proposition is justified by the propositions that directly originated it. The first aspect prevents the formal study of the properties of the systems, independently of their implementations. In those systems, it is very difficult to define and study the properties of the underlying logic except by repeatedly running the program.[5] The second aspect originates systems that can only deal with one situation at a time (and thus are not able to compare two situations) and that present a big computing overhead when switching between situations and when computing the culprit for a contradiction. For a detailed description of these problems see [deKleer 84] and [Martins and Shapiro, forthcoming].

As a reaction to these two problems, the early 80's saw the development of new research directions in belief revision systems, characterized by: (1) an explicit concern with the foundations of the systems, independently of their implementations [Doyle 82, 83; Martins 83; Martins and Shapiro 83] and (2) the use of a new type of justification (giving rise to *assumption-based* belief revision systems) [Martins 83; Martins and Shapiro 83; deKleer 84]. In this paper, we discuss the first issue at length and briefly mention the type of justifications used in the

[5]Although there are techniques to prove properties about programs (and thus one may be tempted to use them to prove properties about theses programs), without the statement of the underlying logic one does not have a clear idea of what properties to prove.

assumption-based belief revision systems. For a detailed description of these systems refer to [Martins and Shapiro, forthcoming].

2. Relevance Logic

The point of this section is to introduce the terminology used by Anderson and Belnap in one of their relevance logic systems and to show how it is used to effectively block some of the results obtainable in classical logic that Anderson and Belnap consider to be irrelevant results. Anderson and Belnap's relevance logic will be taken as the starting point for developing the SWM system. The main features of relevance logic used in SWM are the way it keeps track of which hypotheses were used in the derivation of a given wff and the way this is used to restrict the application of certain rules of inference.

One of the fundamental problems that any logic underlying a belief revision system has to address is how to keep track of and propagate propositional dependencies. This is important, because, in the event of detection of a contradiction, one should be able to identify *exactly which* assumptions were used in the derivation of the contradictory propositions, in order not to blame some assumption irrelevant to the occurrence of the contradiction as the culprit for the contradiction. In the field of logic, the relevance logicians have also been interested in keeping track of what assumptions were used to derive any given proposition; they have developed mechanisms for this, as well as for preventing the introduction of irrelevancies. The SWM system, described in this paper, is based on the relevance logic systems of [Anderson and Belnap 75] and [Shapiro and Wand 76] and thereby makes use of the bookkeeping mechanisms of relevance logic. In addition, SWM has some other mechanisms to remember the contradictions that were derived, to prevent their re-derivation.

Relevance logic was proposed by Anderson and Belnap [Anderson and Belnap 75], reacting to the lack of relevance in classical logic. Relevance logic challenges classical logic in the following two respects:

1. **Regarding the classical concept of validity**: Anderson and Belnap argue that if one proposition entails another, then there must be an element of causality that relevantly connects them. For that reason, they do not recognize as valid some of the arguments classified as valid by classical logic. In particular, they explicitly deny the so-called paradoxes of implication: $A \rightarrow (B \rightarrow A)$, anything implies a true proposition; and $(A \wedge \neg A) \rightarrow B$, a contradiction implies anything. To their (semantic) notion of entailment, corresponds a (syntactic) notion of deducibility, according to which B is deducible from A only if the derivation of B genuinely uses, and does not simply take a detour via, A.

2. **Regarding the applicability of the disjunctive syllogism**: Anderson and Belnap distinguish two kinds of "or", the truth-functional and the intensional. In the *truth functional sense*, the truth value of the formula $A \vee B$ is defined *only* in terms of the truth values of A and B. For this sense of "or", the classical rule of or-introduction is applicable (from A, infer either $A \vee B$ or $B \vee A$, regardless of the truth value of B or its connection with A). *The intensional sense assumes that the disjuncts are relevant to* each other. According to Anderson and Belnap, this corresponds to the way "or" is used in everyday language, meaning something like "if it isn't one, then it is the other". This case clearly involves an entailment (or, better, two entailments), and disjunctive syllogism (from $A \vee B$ and $\neg A$, infer B) can be explained in terms of modus ponens. They argue that classical logic confuses the two senses of "or", using the first in the rule of or-introduction and the second in the rule of or-elimination. For this reason, they consider disjunctive syllogism a "fallacy of relevance" and explicitly deny that it is a valid rule of inference.

We briefly describe how Anderson and Belnap define deducibility in a natural deduction system, the *FR system* [Anderson and Belnap 75, pp.346-348]. Most of this methodology will be adopted in the SWM system. A natural deduction system, e.g. [Fitch 52], contains no axioms, only rules of inference. The rules of inference of a natural deduction system typically contain:

1. A rule of hypothesis, which enables one to get started without the need of axioms from which to begin.

2. Two rules of inference for each logical symbol (logical symbols are either logical connectives or quantifiers), called the introduction and elimination rules. The introduction rule tells how to introduce an occurrence of the logical symbol and is written σI, "σ" being the logical symbol. The elimination rule tells how to eliminate an occurrence of the symbol and is written σE. Some connectives may have more than one elimination rule: for example, $\rightarrow$ has two elimination rules, modus ponens and modus tollens.

In natural deduction systems, a proof is defined to be a nested set of subproofs. A subproof is a list of well-formed formulas (wffs) and/or subproofs. Each wff is contained in a subproof. Subproofs are initiated every time a new hypothesis is introduced (which can be done at any point) and terminated when the hypothesis is discharged. There is one outermost subproof, called "categorical", in which no hypotheses are assumed; the remaining subproofs are called "hypothetical". Theorems are wffs in the categorical subproof.

In FR, to ensure that B is deducible from A only if A is *used* in the derivation of B, Anderson and Belnap restrict the classical rules of natural deduction, as follows:

1. Within a deduction, each wff is associated with a set containing references to all the hypotheses that were *really* used in its derivation. We call this set the *Origin Set* (OS) of the wff and denote the fact that A is a wff with OS α by writing A,α.

2. The rules of inference are stated taking OSs into account, blocking what are considered to be irrelevant applications of the rules allowed in classical logic.

In FR, hypotheses can be introduced at any point (just as in classical natural deduction systems), but whenever a new hypothesis is introduced, it is associated with a singleton OS whose element is an identifier that never appeared before in the proof. Relevance-logic systems typically use natural numbers as elements of the OSs. The rules of inference of the FR system are stated so that all the wffs derived using a particular hypothesis will have its identifier in

their OS. Theorems are wffs with empty OS. When a rule of inference is applied, the resulting wff is associated with an OS which is either the union of the OSs of the parent wffs, the OS of the parent wff(s), or the set difference of the OSs of the parent wffs. To give an idea of how the OSs can be formed, we will elaborate on two rules, $\rightarrow$I and $\wedge$I.

The rule of $\rightarrow$I states that if A is a hypothesis with OS {k}, B is a derived wff with OS $\alpha \cup \{k\}$ (meaning that A was genuinely used in the derivation of B), and they are both in the same subproof, then one can deduce $A \rightarrow B$ (in the subproof immediately containing the subproof initiated by the introduction of the hypothesis A) and associate this new wff with the OS α. This rule is schematically presented in Figure 1. Notice that $A \rightarrow B$ does not depend on hypothesis A. This is the reason for the set-difference operation performed on the OS of B to obtain the OS of $A \rightarrow B$.

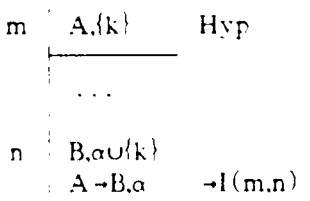


Figure 1
FR system's $\rightarrow$I

The rule of $\wedge$I states that if A and B are wffs with the same OS, then one can deduce $A \wedge B$ and associate it with that OS. This rule is represented in Figure 2.

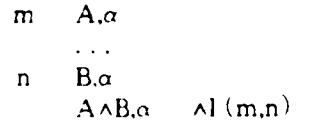


Figure 2
FR system's $\wedge$I

This rule may seem too strongly stated, but it must be so in order to restrict the gratuitous introduction of irrelevancies. Suppose that $\wedge$I allowed the conjunction of wffs with different

OSs, resulting in a wff whose OS was the union of the OSs of the parent wffs. Figure 3 shows how, in this case, we could introduce irrelevancies. The application of ∧E to the wff in line 5, which resulted from such a use of ∧I, allows the hypothesis of line 2 to be "smuggled into" the OS of A (line 6), thereby allowing the "proof" of A→(B→A),one of the paradoxes of implication. The proof makes use of some rules of inference that have not been discussed, namely Repetition, Reiteration, ∧-Elimination, and Modus Ponens. Their full statement can be found in [Martins 83].

1	A,{1}	Hyp
2	B,{2}	Hyp
3	A,{1}	Reit (1)
4	B,{2}	Rep (2)
5	A∧B,{1,2}	∧I (3,4)
6	A,{1,2}	∧E (5)
7	B→A,{1}	MP (2,6)
8	A→(B→A),{}	MP (1,7)

Figure 3
"Proof" in the FR system

3. SW: The System of Shapiro and Wand

Shapiro and Wand designed a logical system based on the FR system of relevance logic to be used as the underlying logic for a question-answering system [Shapiro and Wand 76]. The main difference between FR and the system of Shapiro and Wand (henceforth, the SW system) is that while the former deals with isolated derivations, the latter deals with derivations in which information from other derivations may be taken into account: in the SW system, a wff obtained in a particular derivation may be used later in a different derivation. To obtain this behavior, Shapiro and Wand drop the explicit representation of subproofs (although, conceptually, subproofs still exist in their system) and associate each wff with an extra piece of infor-

mation: the *Origin Tag* (OT). OTs can either be *hyp* or *der*: they distinguish hypotheses from derived wffs. Such a distinction is required by some rules of inference.[6]

Shapiro and Wand envisaged their logic applied to a question-answering system that relied on a knowledge base. The users of this system would enter information into the knowledge base and then use a computer program based on the logic to answer queries about the truth value of specific propositions. The goal of the users is not to prove theorems but rather to know whether a particular proposition is true or false *under some set of assumptions*. We stress "under some set of assumptions", because this qualification introduces a fundamental concept characteristic of the SW system: the notion of an *asserted wff*. In the FR system, within a derivation, all the wffs are asserted. The reason for this is that when one considers some wff within a derivation, it is implicit that one also considers all the hypotheses which were introduced during the derivation (and were not discharged up to the point at which the wff appears). The situation is different in the case of a system relying on a knowledge base, entered by several users with different and even conflicting beliefs. When a user queries the knowledge base, he/she may not want to consider all the information it contains but rather only that information which holds under the specific set of hypotheses that he/she is interested in. Thus, in the SW system, a wff is said to be *asserted* if and only if it belongs to the knowledge base *and* all the hypotheses referenced in its OS are assumed (being considered). This means that a given wff can be in the knowledge base and not be asserted. The wffs in the knowledge base that are not asserted will not be considered for deductions: they will be ignored by the knowledge-base retrieval operations.

In the FR system, when one wants to consider some proposition under some set of hypotheses, a derivation of that proposition has to be carried out assuming those hypotheses. This situation is cumbersome when working with a knowledge base, since, if the proposition

[6]For example, the rule of →I

exists in the knowledge base and was derived from the set of hypotheses under consideration, the user should be able to use it without re-deriving it. This was the main reason that led Shapiro and Wand to design their system without the explicit representation of subproofs: in the SW system, all the asserted wffs are available independently of where and when they were derived.

In the SW system, the fact that the wff A has OT τ and OS α is represented by A,τ,α. We write τ=**ot**(A) and α=**os**(A). The following are the rules of inference allowed in the SW system.[7]

Hypothesis (Hyp): At any point, we may add A,hyp,o to the knowledge base, where A is a wff and o is a singleton set such that no hypothesis of the form B,hyp,o already exists in the knowledge base.

Negation Introduction ($\neg$I): From A,t_1,o, $\neg A,t_2,o$, and any hypothesis H such that **os**(H)$\subset$o, infer $\neg H,der,o-\{H\}$.

Negation Elimination ($\neg$E): From $\neg\neg A,t,o$ infer A,der,o.

And Introduction ($\wedge$I): From A,t_1,o and B,t_2,o, infer $A\wedge B,der,o$.

And Elimination ($\wedge$E): From $A\wedge B,t,o$ infer either A,der,o or B,der,o or both.

Or Introduction ($\vee$I): From A,t,o, infer either $A\vee B,der,o$ or $B\vee A,der,o$, for any proposition B.

Or Elimination ($\vee$E): From $A\vee B,t_1,o_1$, $A\rightarrow C,t_2,o_2$, and $B\rightarrow C,t_3,o_2$, infer $C,der,o_1\cup o_2$.

Implication Introduction ($\rightarrow$I): From B,t,o and any hypothesis H such that **os**(H)$\subset$o, infer $H\rightarrow B,der,o-\{H\}$.

---

[7]We should point out that only the rules of Hyp, $\rightarrow$I, MP, $\wedge$I, and $\wedge$E are actually described in [Shapiro and Wand 76]. The remaining rules reflect our interpretation of how they would handle the other connectives.

Modus Ponens — Implication Elimination, Part 1 (MP): From A,t_1,o_1 and $A \rightarrow B,t_2,o_2$ infer $B,der,o_1 \cup o_2$.

Modus Tollens — Implication Elimination, Part 2 (MT): From $A \rightarrow B,t_1,o_1$ and $\neg B,t_2,o_2$ infer $\neg A,der,o_1 \cup o_2$.

The SW system, records dependencies of propositions but is not prepared to deal with contradictions. We will further modify the SW system, introducing provisos to deal with contradictions and to remember the contradictions that were derived, preventing their re-derivation.

4. The Senses of "or"

There are strong discrepancies between the meaning of the English word "or" and its formal counterpart "$\vee$" (see, for example, [Tarski 65]). Here we discuss two senses of the "or" connective, analyze the rules of $\vee$I and $\vee$E presented in Section 3, and revise those rules in order to obtain certain features to be discussed in this section.

In contemporary logic, it is commonplace to define the rules for $\vee$I and $\vee$E as follows. These rules, according to [Anderson and Belnap 75], define "$\vee$" in the truth-functional sense:

1. **Or Introduction**: From A, infer either $A \vee B$ or $B \vee A$, independently of the truth value of B or its connection with A.
2. **Or Elimination**: From $A \vee B$, $A \rightarrow C$, and $B \rightarrow C$, infer C.

The following can also be given as a rule for Or Elimination, or easily obtained as a derived rule of inference:

3. **Disjunctive Syllogism**: From $A \vee B$ and $\neg A$, infer B; from $A \vee B$ and $\neg B$, infer A.

In relevance logic, if we have a rule corresponding to 1 we cannot have a rule corresponding to 3, since this is forbidden for the truth-functional sense of "or" [Anderson and

Belnap 75, pp.163-167, pp.176-177]. To obtain 3 as a rule for Or Elimination, and to be consistent with relevance logic, we would have to have the following rule for Or Introduction, which introduces "$\vee$" in the intensional sense:

4. **Or introduction**: From $\neg A \rightarrow B$ and $\neg B \rightarrow A$, infer $A \vee B$.

With rule 4, it is possible to introduce $A \vee B$ even if both A and B have unknown truth values, provided that the two entailments can be proven. However, the disjuncts cannot both be false, and this is attractive in the sense that it is close to the way "or" is used in ordinary language.

Unfortunately, neither one of the above alternatives 1 and 2 nor 3 and 4 is suitable for our purpose:

A. Adopting 1 and 2, we lose the possibility of deducing information about one of the disjuncts based on information about the other disjunct. This is one of the features that we want to preserve in our system, since we want to use propositions of the form "one of A and B is true", "at least one of $A_1, \ldots, A_n$ is true", "exactly two of $A_1, \ldots, A_n$ are true", etc., and deduce information about the truth value of some of the arguments based on information about the truth values of the other arguments (see, for example [Martins and Shapiro, forthcoming]). For this reason, we do not want to use 1 and 2.

B. On the other hand, if we adopt 3 and 4 and therefore deny the truth-functionality of the "$\vee$" connective, we can have Disjunctive Syllogism but introduce extra complexity in the rule of Or Introduction.

Instead of opting for one of the alternatives above, we will compromise between them, allowing the co-existence of both senses of "or", but maintaining a clear distinction between them. If "or" is introduced in the truth-functional sense (using 1), then it can only be eliminated using 2, and 3 will not be applicable. If "or" is introduced in the intensional sense (using 4), then it can be eliminated using either 2 or 3. We will distinguish between the two senses of "or" by representing the *truth-functional* sense by "Γ" and the *intensional sense* by

"$\vee$".

Notice that if $A \nabla B$ were introduced because we knew that A was true, and if we could prove that $A \rightarrow C$, there would be no point in trying to derive $B \rightarrow C$ in order to introduce C, since we could get it from A and $A \rightarrow C$. For this reason, we find that rule 2 is useless for the truth-functional sense of "or" and discard it from our collection of rules of inference. This means that we will have a rule to introduce "∇" and no rule to eliminate it.

Thus let us modify SW with the following rules of inference for "or":

Or Introduction, truth-functional sense (∇I): From A,t,o, infer either $A \nabla B$,der,o or $B \nabla A$,der,o, independently of the truth value of B or its connection with A.

Or Introduction, intensional sense ($\vee$I): From $\neg A \rightarrow B, t_1, o$ and $\neg B \rightarrow A, t_2, o$, infer $A \vee B$,der,o.

Or Elimination, intensional sense ($\vee$E):

From $A \vee B, t_1, o_1$ and $\neg A, t_2, o_2$, infer B,der,$o_1 \cup o_2$; from $A \vee B, t_1, o_1$ and $\neg B, t_2, o_2$, infer A,der,$o_1 \cup o_2$.

From $A \vee B, t_1, o_1$, $A \rightarrow C, t_2, o_2$, and $B \rightarrow C, t_3, o_2$, infer C,der,$o_1 \cup o_2$.

Furthermore, we take the position that the sense of "or" used in natural language corresponds to the intensional sense of "or" unless if it is being used in antecedent position of an entailment. Though this may seem strange at first, it is justified by the following:

Claim: In common language, nested wffs in antecedent position of entailments are not propositions of the same importance as the outer propositions: they just represent a syntactic abbreviation for simple, one-level non-nested propositions, and that makes the nested connective truth-functional.

For example, this claim means that we consider the wff $((A \vee B) \wedge C) \rightarrow D$ as an abbreviation for the wff $(A \wedge C) \rightarrow D \wedge (B \wedge C) \rightarrow D$. The disjunction $(A \vee B)$ is not thought of as being an independent wff but as pure economy of expression.

This claim is supported by data gathered by [Rips 83]. Rips developed a psychological theory of propositional reasoning with the goal of explaining the human ability to draw conclusions on the basis of sentence connectives like "and", "if", "or", and "not". Among others, his psychological model has the following rules of inference (adapted to our notation from Table 1 of [Rips 83, p.45]):

1. Disjunctive Modus Ponens: From $(A \vee B) \rightarrow C$ and A, infer C;

2. Or Introduction: From A, infer $A \vee B$.

His theory was implemented in a computer program, called ANDS, and the results supplied by ANDS were successfully compared with those of human subjects untrained in logic.

After completing the experiments, Rips computed the availability of each of the rules of inference, i.e., the probabilities of each of these rules being used (Table 5, [Rips 83, p.62]). He concluded that Disjunctive Modus Ponens is one of the rules with the highest availability (1.000) and that Or Introduction is the rule with lowest availability (0.197).

It is worthwhile noticing that Disjunctive Modus Ponens is, in most classical logic systems, a derived rule of inference obtained by the application of Modus Ponens *following* the application of Or Introduction, and thus the availability of Or Introduction should be at least as high (if not higher) than the availability of Disjunctive Modus Ponens.

What our claim states is that when people say $(A \vee B) \rightarrow C$, they do not mean that the disjunction of A with B entails C; what they really mean is that either A or B entails C and therefore that $(A \vee B) \rightarrow C$ is in fact a syntactic abbreviation for $A \rightarrow C$ and $B \rightarrow C$. This would explain why Or Introduction does not have higher availability than Disjunctive Modus Ponens. We claim that when people state rules with embedded connectives, they are not mentioning the non-atomic propositions from which they are composed, but rather a relationship between the set of atomic propositions.

The co-existence of both senses of "or" and our claim about how to interpret wffs in antecedent position of entailments allow the formalization of natural language's "or" in a way

that is more appropriate than either the classical-logic or the relevance logic-approaches.

5. Extending Origin Tags

The strong constraints imposed upon SW's rule of ∧I motivates the addition of another possible value for the OTs. For the reasons presented in Section 2, the rule of ∧I prevents the conjunction of two wffs with different OSs. Although such a restriction is perfectly justifiable in a logical system whose use is to assess the validity/invalidity of arguments, it may be too constraining in a system designed to support automatic reasoning systems in which the goal is to find out that information holds under some set of assumptions. Recall that in Section 3 we said that a wff was asserted or not in the database depending on whether the hypotheses referenced in its OS were being considered. From this point of view, it should be clear that if both hypotheses A and B are under consideration, then the wff A∧B *should* be assertible in the database since, *given both A and B, clearly* A∧B holds. Using the rules of inference of SW, however, given the hypotheses A and B, we cannot assert A∧B, since A and B have different OSs. In this section, we discuss what we should do in order to be able to assert the wff A∧B from the wffs A and B.

Simply dropping the constraint on the rule of ∧I would let us assert A∧B from A and B, but, as discussed in Section 2, would also open a door to the introduction of irrelevancies. Let us consider again the derivation presented in Figure 3. We have seen that in this derivation it is not possible to assert the conjunction A∧B,{1,2}. Now, suppose that we re-do the derivation by changing the hypotheses as shown in Figure 4.

1	A ∧ (B→B),{1}	Hyp
2	A,{1}	∧E (1)
3	B→B,{1}	∧E (1)
4	B ∧ (A→A),{2}	Hyp
5	B,{2}	∧E (4)
6	B→B,{1}	Reit (3)
7	B,{1,2}	MP (5,6)
8	A,{1}	Reit (2)
9	A→A,{2}	∧E (4)
10	A,{1,2}	MP (8,9)
11	A∧B,{1,2}	∧I (7,10)

Figure 4
Partial derivation in the FR system

By conjoining to each hypothesis of the derivation a wff that is a theorem in the FR system, (P→P), we were able to derive A∧B,{1,2} within the inner subproof (line 11). The rule of →I allows us to bring A∧B out of the inner subproof, resulting in the entailment [B ∧ (A→A)]→(A∧B),{1}. But we are not able to derive either B→(A∧B),{1}, or B→A,{1} since these are not relevant implications.

What we have shown is that by conjoining theorems to the hypotheses of a proof, we may "extend" the OS of wffs, thereby being able to conjoin wffs that were not conjoinable in the original proof. Instead of going through all the trouble of deriving a new proof with slightly changed hypotheses, we will allow the application of ∧I among wffs with different OSs, provided that the resulting wff is "marked" as a special wff. To mark wffs as special, we introduce a new value (*ext*) for the OTs. A wff whose OT is *ext* is neither a hypothesis nor a normally derived wff but rather a special wff, which, if it were treated regularly, would introduce irrelevancies into the knowledge base. We should stress again that the reason why we don't want to introduce irrelevancies in the knowledge base is that, after the detection of a contradiction, we don't want to blame the contradiction on some proposition that is *irrelevant* to the contradiction. This new OT is introduced when we conjoin two wffs with different OSs. The rule of ∧I thus becomes:

And Introduction ($\wedge$I):

From A,t_1,o and B,t_2,o, where $t_1 \neq$ext and $t_2 \neq$ext, infer A$\wedge$B,der,o; from A,t_1,o and B,t_2,o, where t_1=ext, or t_2=ext, infer A$\wedge$B,ext,o; from A,t_1,o_1 and B,t_2,o_2, where $o_1 \neq o_2$, infer A$\wedge$B,ext,$o_1 \cup o_2$.

The inference rules of SW have to be modified to cope with this new OT: some of them will not apply to wffs with this OT (namely, the rules of $\neg$I and $\wedge$E), and those which still apply must mark the resulting wff with the special OT as well. These new rules of inference will be presented in Section 10.

6. Restriction Sets and Belief Revision

The SW system allows derivations in which information from other derivations may be taken into account. In belief revision systems, one type of information that is important to share among derivations concerns the conditions under which contradictions may occur. With the goal of recording the conditions under which contradictions occur, we further associate each wff with a set called the *Restriction Set* (RS).

An RS is a set of sets of wffs. Having a wff, say A, whose RS is $\{R_1, \ldots, R_n\}$ means that the hypotheses in the OS of A added to any of the sets $R_1, \ldots, R_n$ produces an *inconsistent set*, i.e., a set from which a contradiction may be derived[6]. The role of RSs is thus to record the conditions under which contradictions occur.

The rules of inference of the SW system will be refined in Section 10 to cope with the addition of RSs. The application of rules of inference is blocked if the resulting wff would have an OS known to be inconsistent. In other words, RSs allow the recording of which sets are *known* to be inconsistent and prevent the derivation of wffs with OSs that are *known* to be inconsistent.

[6]A set is consistent just in case it is not inconsistent, i.e., no contradiction may be proved in it.

It is important to distinguish between a set *being* inconsistent and a set *being known to be* inconsistent. An inconsistent set is one from which a contradiction *can be* derived; a set known to be inconsistent is an inconsistent set from which a contradiction *has been* derived. The goal of adding RSs is to avoid re-considering *known* inconsistent sets of hypotheses.

RSs can be introduced when entering hypotheses, having in mind the idiosyncrasies of the domain being modeled, but mainly they are introduced after detecting a contradiction. Upon finding a contradiction, the rules of inference of our logic (described in Section 10) allow:

1. The modification of the RSs of the hypotheses underlying the contradiction (and the RSs of the wffs derived from them) in a way that records the occurrence of the contradiction, preventing its repetition.
2. The derivation of new wffs whose RSs reflect the occurrence of the contradiction.

It should be pointed out that RSs are very different entities from OTs and OSs. Whereas the OT and OS of a proposition reflect the way the proposition was derived, the RS of a proposition reflects our current knowledge about how the hypotheses underlying that proposition relate to the other hypotheses in the knowledge base. Once a proposition is derived, its OT and OS remain constant, whereas its RS changes as the knowledge about all the propositions in the knowledge base does.

7. The SWM system

The logic resulting from FR by dropping the explicit representation of the structure of subproofs, associating each wff with an OT and an RS as well as an OS, and modifying the rules of inference is called the *SWM System*.[9]

[9] After Shapiro, Wand, and Martins.

Each wff in SWM is associated with an OT, an OS, and an RS and is called a *supported wff*. We write $A \mid \tau,\alpha,\rho$ to denote that A is a wff with OT τ, OS α, and RS ρ, and we define the functions $ot(A)=\tau$, $os(A)=\alpha$, and $rs(A)=\rho$.[10] The OSs are sets of hypotheses. The OTs range over the set {hyp,der,ext}: *hyp* identifies hypotheses, *der* identifies normally derived wffs within SWM, and *ext* identifies wffs whose OS was extended. The RSs are sets of sets of hypotheses. The set of all supported wffs is called the *knowledge base*.

The rules of inference of SWM will be presented in Section 10.

8. Computing RSs

Let us recall the definition of an RS: having the supported wff $A \mid \tau,\alpha,\{R_1,\ldots,R_n\}$ means that the hypotheses in α added to any of the sets $R_1,\ldots,R_n$ produce an inconsistent set, a set from which a contradiction can be derived, i.e., $\forall\rho\in\{R_1,\ldots,R_n\}(\alpha\cup\rho \vdash \rightarrow\leftarrow)$.[11] In this section, we analyze what the RS of a supported wff resulting from the application of some rule of inference should be. Our goal is to compute the resulting RS, keeping it as small as possible, i.e., free of redundancies.

Let us start by describing the kinds of redundancies that may be present in RSs and discuss how they can be eliminated.

1. Suppose that in $A \mid \tau,\alpha,\{R_1,\ldots,R_n\}$, $\exists\rho\in\{R_1,\ldots,R_n\}$ such that $\rho\cap\alpha\neq\emptyset$. In this case, the set ρ contains extra information, namely, all the wffs in $\rho\cap\alpha$. Let Ψ be defined as:

$$\Psi(R,O) = \{\alpha : (\alpha\in R \wedge \alpha\cap O=\emptyset) \vee (\exists\beta\in R)[\beta\cap O\neq\emptyset \wedge \alpha=\beta-O]\}$$

These redundancies will be eliminated by letting the RS of A be $\Psi(\{R_1,\ldots,R_n\},\alpha)$ instead of $\{R_1,\ldots,R_n\}$.

---

[10] We do not consider here the problem of multiple derivations of the same proposition; this is discussed in [Martins and Shapiro forthcoming].

[11] $\rightarrow\leftarrow$ represents a contradiction and $\vdash$ represents deducibility.

2. Suppose that in the RS of $A \mid \tau,\alpha,\{R_1, \ldots, R_n\}$ there exist sets, say μ and ν, such that $\mu \subset \nu$. We claim that ν can be discarded from $\{R_1, \ldots, R_n\}$ without any loss of information. Why? The fact that μ belongs to the RS means that $\alpha \cup \mu \vdash \rightarrow\leftarrow$. Also, using the fact that any set containing an inconsistent set is itself inconsistent (Theorem 4, in Appendix 1), we can infer that $\alpha \cup \nu$ is inconsistent, since $(\alpha \cup \mu) \subset (\alpha \cup \nu)$. Therefore, the inclusion of ν in the RS is unnecessary. We denote by $\sigma(R)$ the set obtained from R by discarding this kind of redundant set:

$$\sigma(R) = \{ \alpha \in R \mid \neg(\exists \beta)(\beta \neq \alpha \wedge \beta \in R \wedge \beta \subset \alpha) \}.$$

3. Suppose that in the RS of $A \mid \tau,\alpha,\{R_1, \ldots, R_n\}$ there exists a set, say ρ, such that from a proper subset of ρ, together with α, we can derive a contradiction, i.e.,

$$\exists \rho \in \{R_1, \ldots, R_n\} : \exists \nu \subset \rho : \nu \neq \rho \wedge \nu \cup \alpha \vdash \rightarrow\leftarrow.$$

In this case, all the wffs in $\rho - \nu$ are irrelevant to the contradiction and therefore do not need to be stored in the RS. There are basically two ways of handling this case:

a. Whenever an RS is created, we will try to prove that subsets of its sets, together with the OS of the wff can yield a contradiction. Clearly this case is not practical.

b. We do not worry about these cases, and let them be handled in practice when they are discovered. In other words, when we build a new RS we ignore the possibility of the occurrence of such a case and proceed normally. If later on we find out that there exists such a subset s, we add s to the RS and let the function σ take care of it.

Having in mind the points just discussed, we will say that the supported wff $A \mid \tau,\alpha,\{R_1, \ldots, R_n\}$ has a *minimal RS* if the following two conditions are met:

1. $\forall r \in \{R_1, \ldots, R_n\}\ (r \cap \alpha) = \emptyset$
2. $\forall r,s \in \{R_1, \ldots, R_n\}\ r \not\subset s$.

Now, let us consider what the RS of a supported wff resulting from the application of some rule of inference should be. We will assume that we are applying rules of inference to sup

ported wffs that have minimal RSs and that our goal is to compute a minimal RS for the resulting supported wff. In Appendix 1, we show that all the supported wffs in the knowledge base resulting from the application of the rules of inference of SWM have minimal RSs and therefore that our assumption holds.

To compute the RS of a supported wff resulting from the application of the rules of inference of SWM, we will consider the rules of inference of SW. The reason for this is that SWM will have basically the same set of rules of inference as SW except that (a) the wffs will have an RS, and (b) some of the applications of the rules of inference will be blocked.

An analysis of the rules of inference of SW shows that they can be applied to either one wff or two wffs. The rules of inference that are applied to only one wff ($\neg$E, $\wedge$E, and ∇I) result in a supported wff that has the same OS (and therefore should have the same RS) as the parent wff. The remaining rules of inference require two supported wffs for their application. Suppose that we are given two supported wffs to be combined by some rule of inference and that we want to compute a minimal RS for the resulting supported wff. Our approach will be to look at the RSs of the parent wffs and determine which sets added to the OS of the resulting wff will produce an inconsistent set. An inspection of the rules of inference of the SW system shows that the OS of the resulting wff can either be the union of the OSs of the parent wffs[12] or the set difference of the OSs of the parent wffs. Let us consider each of these cases in turn:

1. Suppose that we are given $A \mid t_a, o_a, \{R_1, \ldots, R_n\}$ and $B \mid t_b, o_b, \{S_1, \ldots, S_m\}$ to be combined by some rule of inference and that the OS of the resulting wff is $o_a \cup o_b$. (This corresponds to the application of the rules MP, MT, $\wedge$I, $\vee$I, and $\vee$E): From the definition of RS, we know that $\forall r \in \{R_1, \ldots, R_n\}\ o_a \cup r \vdash \rightarrow\leftarrow$ and $\forall s \in \{S_1, \ldots, S_m\}\ o_b \cup s \vdash \rightarrow\leftarrow$. Using the fact that any set containing an inconsistent set is itself inconsistent (Theorem 4, in

[12]This includes the case in which the parent wffs have the same OS.

Appendix 1), we can infer that

$$\forall r \in \{R_1, \ldots, R_n\}\ (o_a \cup o_b) \cup r \vdash \rightarrow\leftarrow \text{and}$$

$$\forall s \in \{S_1, \ldots, S_m\}\ (o_a \cup o_b) \cup s \vdash \rightarrow\leftarrow.$$

In other words, adding any one of $R_1, \ldots, R_n, S_1, \ldots, S_m$ to $o_a \cup o_b$ produces an inconsistent set. By the definition of RS, it is clear that the RS of the resulting wff should record the information contained in $\{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$. However, if we let the resulting RS be $\{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$, the resulting wff may not have a minimal RS. To generate a supported wff with minimal RS, we argue that two simplifications may have to be performed on $\{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$. In the the RS of C $t_c, o_a \cup o_b, \{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$, there may be an r such that $r \not\subset (o_a \cup o_b) \neq C$,[13] and thus Ψ must be applied to the union of the RSs of the parent wffs. Furthermore there may also exist $r,s \in \{R_1, \ldots, R_n\} \cup \{S_1, \ldots, S_m\}$ such that $r \subset s$[14] and, therefore, σ must also be applied to the resulting RS. We denote by μ the operation

$$\mu(\{r_1, \ldots, r_m\}, \{o_1, \ldots, o_n\}) = \sigma(\Psi(r_1 \cup \ldots \cup r_m, o_1 \cup \ldots \cup o_n)),$$

and define the RS of the resulting wff to be $\mu(\{\{R_1, \ldots, R_n\}, \{S_1, \ldots, S_m\}\}, \{o_a, o_b\})$. Notice that the order of application of the operations σ and Ψ is not irrelevant.[15] We prove in Appendix 1 that for every two sets A and B we have $\sigma(\Psi(A,B)) \subset \Psi(\sigma(A),B)$ (Theorem 1) and that given $A \mid t_a, o_a, r_a$ and $B \mid t_b, r_b, o_b$, the proposition $C \mid t_c, o_a \cup o_b, \mu(\{r_a, r_b\}, \{o_a, o_b\})$ has, in fact, minimal RS (Theorem 2).

2. Suppose that we are given $A \mid t_a, o_a, \{R_1, \ldots, R_n\}$ and $B \mid t_b, o_b, \{S_1, \ldots, S_m\}$ to be combined by some rule of inference and that the OS of the resulting wff is $o_a - o_b$. (This

[13]Consider, for example, the supported wffs A | hyp,{A},{{A→C,B→¬C,B}} and A→C | hyp,{A→C},{{B→¬C,A,B}}. Applying the rule of MP to them, we obtain C | der,{A,A→C},R. If we let R be {{A→C,B→¬C,B},{B→¬C,A,B}} (the union of the two RSs), C would not have a minimal RS, since the members of its RS are not disjoint from its OS.

[14]Consider, for example, the supported wffs D | hyp,{D},{{A}} and D→P | hyp,{D→P},{{A,B}}.

[15]For example, $\sigma(\Psi(\{\{A,B,C\},\{A,D\}\},\{D,E\})) = \sigma(\{\{A,B,C\},\{A\}\}) = \{\{A\}\}$ and $\Psi(\sigma(\{\{A,B,C\},\{A,D\}\}),\{D,E\}) = \Psi(\{\{A,B,C\},\{A,D\}\},\{D,E\}) = \{\{A,B,C\},\{A\}\}$.

corresponds to applications of $\neg$I and $\neg$I): To form the RS of the resulting wff, we want to find out which sets added to $o_a - o_b$ will produce an inconsistent set. At first it seems that knowing that the n sets (involving o_a) $o_a \cup r$, for $r \in \{R_1, \ldots, R_n\}$ are inconsistent it suffices to find sets $\{X_1, \ldots, X_n\}$ such that

$$\forall r \in \{R_1, \ldots, R_n\}\ \exists x \in \{X_1, \ldots, X_n\} : x \cup (o_a - o_b) = o_a \cup r.$$

Such sets would be obtained by solving each of the n equations above, obtaining $x = r \cup o_b$ for each r in $\{R_1, \ldots, R_m\}$ and therefore concluding that the RS of the resulting wff should be $\{R_1 \cup o_b, \ldots, R_m \cup o_b\}$. However, things are not as easy as they initially seem. The reasoning just described would be correct if $\{S_1, \ldots, S_m\} = \{\}$. Consider the following example: suppose that we combine A | der,{B,C},{{F,G},{M,N}} with K | hyp,{K},{{M}} resulting in the supported wff J | der,{B,C,K},{{F,G},{M}}. From this latter supported wff we can derive K$\rightarrow$J under the OS {B,C}. But what should be the RS of this wff? According to the reasoning just outlined, it should be {{F,G,K},{M,K}}. However, there are two problems with this RS:

1. it fails to record the inconsistent set {B,C,M,N};

2. it contains a set ({M,K}) which is known to be inconsistent.

The reason behind these two problems is that when some supported wff with non-empty RS is combined with some other supported wff, it results in a supported wff from whose RS we may not compute the RSs of the parent supported wffs. The only way that we see to solve this type of problem is to compute the RS of the supported wff whose OS is $o_a - o_b$ directly from the RSs of the hypotheses in $o_a - o_b$.[16] Denoting by the operation

$$(O) = \mu(\{r : \exists H \in O : r = \mathbf{rs}(H)\}, \{o : \exists H \in O : o = \mathbf{os}(H)\})$$

we conclude that the RS of the resulting wff should be $(o_a - o_b)$. We have proved

[16]The problem of computing this RS can be avoided in computer implementations of this logic, see, for example [Martins and Shapiro, forthcoming].

(Theorem 3, in Appendix 1) that $C \mid t_c, o_a - o_b, (o_a - o_b)$ has a minimal RS.

9. Conditions of Combinability of Wffs

Two supported wffs are *combinable* by some rule of inference if the supported wff resulting from the application of the rule of inference has an OS that is not known to be inconsistent. In this section we define the conditions under which two supported wffs are combinable.

To do this, we turn again to the two ways of forming the OS of the wff resulting from combining $A \mid t_a, o_a, \{R_1, \ldots, R_n\}$ and $B \mid t_b, o_b, \{S_1, \ldots, S_m\}$:

1. The OS of the resulting wff is $o_a \cup o_b$: To make sure that $o_a \cup o_b$ is not known to be inconsistent, we will require that it does not contain any of the inconsistent sets involving either o_a or o_b that we know of: $o_a \cup r$ for $r \in \{R_1, \ldots, R_n\}$ and $o_b \cup s$ for $s \in \{S_1, \ldots, S_m\}$. In other words, we want to guarantee that $\forall r \in \{R_1, \ldots, R_n\}\ (o_a \cup r) \in (o_a \cup o_b)$ and $\forall s \in \{S_1, \ldots, S_m\}\ (o_b \cup s) \in (o_a \cup o_b)$. Since r is disjoint from o_a and s is disjoint from o_b, this entails that the following conditions have to be met:

   $\forall r \in \{R_1, \ldots, R_n\}\ r \subset o_b$, and

   $\forall s \in \{S_1, \ldots, S_m\}\ s \subset o_a$.

   This means that the RS of one of the wffs cannot contain a set contained in the OS of the other wff. We define the predicate *Combine*, which decides the combinability of the supported wffs A and B:

$$\text{Combine}(A,B) = \begin{cases} \textit{false} & \text{if } \exists r \in \mathbf{rs}(A) : r \subset \mathbf{os}(B) \\ \textit{false} & \text{if } \exists r \in \mathbf{rs}(B) : r \subset \mathbf{os}(A) \\ \textit{true} & \text{otherwise} \end{cases}$$

2. The OS of the resulting wff is $o_a - o_b$: In this case, since o_a is not known to be an inconsistent set (the rules of inference will be stated in a way that will prevent the

derivation of a wff with an OS known to be inconsistent), we can conclude that neither is $o_a - o_b$ (see Corollary 4.1 in Appendix 1).

10. Rules of Inference of SWM

We now present the rules of inference of SWM. To make the rules easier to state, we use the function Λ defined on OTs as follows:

$$\Lambda(a,b) = \begin{cases} \text{ext} & \text{if } a=\text{ext or } b=\text{ext} \\ \text{der} & \text{otherwise} \end{cases}$$

This function takes as arguments the OTs of wffs to be combined by some rule of inference and produces the OT of the resulting wff.

Hypothesis (Hyp): For any wff A and sets of wffs $R_1, \ldots R_n$ $(n \geq 0)$, such that $\forall r \in \{R_1, \ldots, R_n\}$: $r \sigma\{A\} = \Sigma$ and $\forall r,s \in \{R_1, \ldots, R_n\}$: $r \subset s$, we may add the supported wff $A \mid hyp,\{A\},\{R_1, \ldots, R_n\}$ to the knowledge base, provided that A has not already been introduced as a hypothesis.

Negation Introduction (¬I): From $A \mid t_1,o,r$, $\neg A \mid t_2,o,r$, and any set $\{H_1, \ldots, H_n\} \subset o$, infer $\neg(H_1 \wedge \ldots \wedge H_n) \mid \Lambda(t_1,t_2),o-\{H_1, \ldots, H_n\},(o-\{H_1, \ldots, H_n\})$. From $A \mid t_1,o_1,r_1$, $\neg A \mid t_2,o_2,r_2$, $o_1 \neq o_2$, Combine(A,¬A), and any set $\{H_1, \ldots, H_n\} \subset (o_1 \cup o_2)$, infer $\neg(H_1 \wedge \ldots \wedge H_n) \mid ext,(o_1 \cup o_2)-\{H_1, \ldots, H_n\},((o_1 \cup o_2)-\{H_1, \ldots, H_n\})$.

Negation Elimination (¬E): From $\neg\neg A \mid t,o,r$, infer $A \mid \Lambda(t,t),o,r$.

And Introduction (∧I): From $A \mid t_1,o,r$ and $B \mid t_2,o,r$, infer $A \wedge B \mid \Lambda(t_1,t_2),o,r$. From $A \mid t_1,o_1,r_1$, $B \mid t_2,o_2,r_2$, $o_1 \neq o_2$, and Combine(A,B), infer $A \wedge B \mid ext,o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

And Elimination (∧E): From $A \wedge B \mid t,o,r$, and $t \neq ext$, infer either $A \mid der,o,r$ or $B \mid der,o,r$ or both.

Or Introduction (truth functional) (∨I): From $A \mid t,o,r$, infer either $A \vee B \mid \Lambda(t,t),o,r$ or

$B \nabla A \mid \Lambda(t,t),o,r$, for any wff B.

Or Introduction (intensional) ($\vee$I): From $\neg A \rightarrow B \mid t_1,o,r$ and $\neg B \rightarrow A \mid t_2,o,r$, infer $A \vee B \mid \Lambda(t_1,t_2),o,r$.

Or Elimination ($\vee$E):

a. From $A \vee B \mid t_1,o_1,r_1$, $\neg A \mid t_2,o_2,r_2$, and Combine($A \vee B, \neg A$), infer $B \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$. From $A \vee B \mid t_1,o_1,r_1$, $\neg B \mid t_2,o_2,r_2$, and Combine($A \vee B, \neg B$), infer $A \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

b. From $A \vee B \mid t_1,o_1,r_1$, $A \rightarrow C \mid t_2,o_2,r_2$, $B \rightarrow C \mid t_3,o_2,r_2$, and Combine($A \vee B, A \rightarrow C$), infer $C \mid \Lambda(t_1,\Lambda(t_2,t_3)),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

Implication Introduction ($\rightarrow$I): From $B \mid der,o,r$ and any hypothesis $H \in o$, infer $H \rightarrow B \mid der,o-\{H\},(o-\{H\})$.

Modus Ponens — Implication Elimination, Part 1 (MP): From $A \mid t_1,o_1,r_1$, $A \rightarrow B \mid t_2,o_2,r_2$, and Combine($A, A \rightarrow B$), infer $B \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

Modus Tollens — Implication Elimination, Part 2 (MT): From $A \rightarrow B \mid t_1,o_1,r_1$, $\neg B \mid t_2,o_2,r_2$, and Combine($A \rightarrow B, \neg B$), infer $\neg A \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

Updating of Restriction Sets (URS): From $A \mid t_1,o_1,r_1$, and $\neg A \mid t_2,o_2,r_2$, we *must* replace each hypothesis $H \mid hyp,\{H\},R$ such that $H \in (o_1 \cup o_2)$ by $H \mid hyp,\{H\},\sigma(R \cup \{(o_1 \cup o_2)-\{H\}\})$. Furthermore, we *must* also replace every supported wff $F \mid t,o,r$ (t=der or t=ext) such that $o \cap (o_1 \cup o_2) \neq \emptyset$ by $F \mid t,o,\sigma(r \cup \{(o_1 \cup o_2)-o\})$.

$\forall$ Introduction ($\forall$I): From $B(t) \mid der,o \cup \{A(t)\},r$, in which $A(t)$ is a hypothesis that uses a term (t) never used in the system prior to A's introduction, and t is not in o or r, infer

$\forall(x)[A(x)\rightarrow B(x)] \mid der,o,(o)$.[17]

∀ Elimination - Universal Instantiation (∀E): From the supported wffs $\forall(x)[A(x)\rightarrow B(x)] \mid t_1,o_1,r_1$, $A(c) \mid t_2,o_2,r_2$, and Combine($\forall(x)[A(x)\rightarrow B(x)]$,A(c)), where c is any individual symbol, infer $A(c)\rightarrow B(c) \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

∃ Introduction (∃I): From $A(c) \mid t,o,r$ where c is an individual constant, infer $\exists(x)[A(x)] \mid \Lambda(t,t),o,r$.

∃ Elimination (∃E): From $\exists(x)[A(x)] \mid t,o,r$ infer $A(c) \mid \Lambda(t,t),o,r$ where c is any individual constant that was never used before.

The following theorem holds for SWM (its proof can be found in Appendix 1):

Theorem 5:[18] All supported wffs in the knowledge base resulting from the application of the rules of inference of SWM have minimal RS.

The rules of ¬I (part 1), ∧I (part 1) and ∨I are only applicable to supported wffs that have the same OS and the same RS. This condition is not as constraining as it may seem at first glance, since, if two supported wffs have the same OS, then they also have the same RS, as the following theorem states (its proof can be found in the Appendix 1):

Theorem 6: In the knowledge base resulting from the application of the rules of inference of SWM, if two supported wffs have the same OS, then they have the same RS as well.

Corollary 6.1: Every OS has recorded with it *every* known inconsistent set.

---

[17]According to this rule of inference, the universal quantifier can only be introduced in the context of an implication. This is not a drawback as it may seem at first since the role of the antecedent of the implication (A(x)) is to define the type of objects which are being quantified. This is sometimes called *relativized quantification*.

[18]The numbers of the theorems correspond to the numbers presented in Appendix 1.

11. A Contextual Interpretation for SWM

Having presented the SWM system, we now discuss how a computer program using SWM should interpret SWM's wffs and how SWM's features can be used effectively in applications of belief revision. In this section, we provide what we call a *contextual interpretation* for SWM. We use the phrase "contextual interpretation" instead of just "interpretation" for the following two reasons: on the one hand, we want to stress that we are not providing a full-fledged interpretation for SWM in the logician's sense of the word; on the other hand, we want to emphasize that our definition of truth depends on the notion of context. This contextual interpretation defines the behavior of an abstract belief revision system (abstract in the sense that it is not related to any particular implementation) which we call MBR (the **M**ultiple **B**elief **R**easoner).

MBR works with a knowledge base containing propositions associated with an OT, OS, and RS (in SWM's sense). The propositions that are added to this knowledge base follow the rules of inference of SWM. Within this knowledge base, there may exist contradictory wffs because different users with conflicting interests may have entered contradictory information. When one of the users queries the knowledge base, he/she may not care what the interests of the other users are. Among all the wffs in the knowledge base, there is, for each user, a set of distinguished ones, called the set of *asserted wffs*, which contains those and only those wffs that that user wants MBR to consider.

We assume that each user of the knowledge base has some primitive set of propositions that he/she wants to consider and that he/she told the system about. Such propositions were entered into the knowledge base as hypotheses. Every proposition derived from this set of assumptions will be considered by that particular user. Each user considers all the propositions in the closure of his/her set of assumptions under the rules of inference. We are also assuming that rules of inference are meta-level entities that are accepted by every user of the system. This latter assumption seems reasonable since the rules of inference are the entities that guide

the reasoning of the system.[19]

We define a *context* to be a set of hypotheses, representing the set of primitive assumptions of some user. A context determines a *Belief Space* (BS), which is the set of all the hypotheses defining the context and all the propositions that have been derived exclusively from them. Within SWM the wffs in a given BS are characterized by having an OS that is contained in the context. It follows from this definition that the set of contexts represented in the knowledge base at any time is the power set of the set of hypotheses existing in the knowledge base.

Any operation performed within the knowledge base (query, addition, deletion, etc.) will be associated with a context. We will refer to the context under consideration, i.e., the context associated with the operation currently being performed in the knowledge base, as the *current context*. While the operation is being carried out, the only propositions that will be considered are the propositions in the BS defined by that context. A proposition is said to be *believed* if it belongs to the BS under consideration (the BS defined by the context under consideration, also called the *current belief space*). We can look at contexts as delimiting smaller knowledge bases, the Belief Spaces, within the knowledge base. The only propositions in the knowledge base that are retrievable are those propositions that belong to the current BS.

12. Consistent vs. Inconsistent Contexts

A common goal among knowledge-base users is to stay away from contradictions, i.e., to avoid the simultaneous belief of a proposition and its negation. Taking this into account, it would seem natural to constrain contexts to be consistent sets of hypotheses, not just any sets of hypotheses. Let us note, however, that determining whether a contradiction is derivable from a set of hypotheses is a difficult problem in logic, and thus the condition that contexts are

[19] [Konolige 85] presents a system in which different users can have different rules of inference.

not inconsistent may be very difficult to enforce. For that reason, we may settle for the weaker condition that contexts are not known to be inconsistent.

Within MBR, we can easily detect whether a context is not known to be inconsistent by considering the RSs of the hypotheses defining the context. Given the context $\{H_1, \ldots, H_n\}$, the condition

$$\forall H \in \{H_1, \ldots, H_n\}\ \forall r \in rs(H) : r \notin \{H_1, \ldots, H_n\} - \{H\}$$

guarantees that this context is not known to be inconsistent[20].

However, it may be the case in MBR that one desires to perform reasoning within the BS defined by an inconsistent context, a kind of counterfactual reasoning. This can be done in MBR, because the existence of contradictions in SWM is not as damaging as in classical logic, in which anything can be derived from a contradiction. Also due to this, in MBR one may not want to bother discarding hypotheses after a contradiction is detected, since not the entire system will be affected by the contradiction.

In MBR, the condition that a context is not known to be inconsistent will not be compulsory but rather advisable if one doesn't explicitly want to perform reasoning in a BS that is known to be inconsistent. The reason it is advisable is that within a BS defined by a context not known to be inconsistent, some simplification can be considered during the application of the rules of inference, as stated by the following theorems (their proofs can be found in the Appendix 1):

Theorem 7: If C is a context that is not known to be inconsistent, then, for any two wffs, A and B, in the BS defined by the context C, we have Combine(A,B)=true.

Corollary 7.1: If one uses a context that is not known to be inconsistent, then MBR does not need to check for combinability between the wffs before the application of rules of inference.

[20] The condition $\exists H \in \{H_1, \ldots, H_n\}\ (\exists r \in rs(H) : (\{H\} - r) \subseteq \{H_1, \ldots, H_n\}$ guarantees that the context $\{H_1, \ldots, H_n\}$ is *known* to be inconsistent

13. Two levels of Belief Revision

Let us now consider how MBR acts when a contradiction is detected. We will discuss two levels of belief revision: belief revision within the current context and belief revision within a context strictly containing the current context. The main difference between them is that the former may require changes in the current context and allows the deduction of new wffss while the latter leaves this context unchanged and does not allow the deduction of new wffs to the knowledge base.

SWM has two rules of inference to handle contradictions: negation introduction ($\neg$I) and updating of restriction sets (URS).

The rule of $\neg$I states that from the combinable supported wffs $A \mid t_1,o_1,r_1$ and $\neg A \mid t_2,o_2,r_2$, we can deduce the negation of the conjunction of any number of hypotheses in $o_1 \cup o_2$ under an OS containing the remaining hypotheses. This rule is applied whenever two contradictory wffs are found within the current BS. Its effect is twofold: (1) It may cause the current context to be changed. The fact that both A and $\neg A$ were derived within the current BS means that the current context is *now known* to be an inconsistent set. If one wants to maintain contexts that are not known to be inconsistent, then the current context has to be changed. (2) It allows the addition of new wffs to the knowledge base. Such wffs are negations of conjunctions whose conjuncts are hypotheses in the current context (the hypotheses in $o_1 \cup o_2$).

The rule of URS has the effect of recording the occurrence of contradictions in the RSs of all the hypotheses underlying a contradiction (and the wffs derived from them). This rule, however, does not allow the addition of new wffs to the knowledge base. This rule is obligatorily applied whenever two contradictory wffs are found, whether or not they belong to the current BS. Upon application of this rule, there will be an explicit record in the knowledge base about the possibility of the derivation of the contradictory wffs.

When a contradiction is detected, one of two things will happen:

1. *Only one of the contradictory wffs belongs to the current BS.*[21] The contradiction is recorded (through the application of URS), but nothing more happens. The effect of doing so is to record that some set of hypotheses, properly containing the current context, is now known to be inconsistent. This results in what we call *belief revision within a context properly containing the current context*. This type of revision of beliefs has the effect of recording that a BS larger than the current BS is inconsistent.

2. *Both contradictory wffs belong to the current BS.* URS is applied, resulting in the updating of the RSs of the hypotheses in the current context (and the derived wffs in the current BS). In addition, the rule of ¬I may also be applied. This results in what we call *belief revision within the current context*.

14. An Annotated Example

In order to clarify the concepts introduced so far, we present an example obtained using a particular implementation of MBR. The system whose output we show here, called SNeBR (SNePS Belief Revision), is an implementation of MBR using the SNePS semantic network processing system [Shapiro 79a][22] and is written in Franz Lisp [Foderaro, Sklower and Layer 84], running on VAX-11 systems at the Department of Computer Science of the State University of New York at Buffalo and at the Instituto Superior Tecnico (School of Engineering of the Technical University of Lisbon, Portugal). The results shown in this example represent a slightly edited version of the output generated by SNeBR when it uses a programming interface called SNePSLOG [McKay and Martins 81], a logic programming interface to SNePS.

[21]Note that at least one of the contradictory wffs belongs to the current BS, since a contradiction is detected whenever some newly derived wff contradicts some existing one, and newly derived wffs always belong to the current BS.

[22]In SNeBR, propositions are represented by SNePS network nodes, each proposition being linked with the hypotheses in its OS and the sets in its RS.

In this example, SNeBR is used as a meeting-scheduling system. The knowledge base contains general statements reflecting policies for scheduling meetings and also statements concerning the particular schedules of the SNeBR users.

SNeBR is asked to schedule meetings among a certain number of its users; it does so either by finding a time slot that is compatible with their particular schedules or by reporting that the schedules of the users do not allow the scheduling of the desired meeting. In this example, we will assume that:

1. Meetings are being scheduled within one day only; therefore, information about dates is absent from our representation.

2. Meetings cannot both be in the morning and in the afternoon. This assumption is represented by the following proposition:

   $\forall(x)[\mathbf{meeting}(x) \rightarrow (\mathbf{time}(x,\mathbf{morning}) \rightarrow \neg\, \mathbf{time}(x,\mathbf{afternoon}))]$

3. Two different meetings cannot fill the same time slot, e.g., morning or afternoon. This assumption is represented by the following proposition:

$$\forall(x,y)[(\mathbf{meeting}(x) \wedge \mathbf{meeting}(y) \wedge x \neq y) \rightarrow ((\mathbf{time}(x,\mathbf{morning}) \rightarrow \mathbf{time}(y,\mathbf{afternoon})) \wedge (\mathbf{time}(x,\mathbf{afternoon}) \rightarrow \mathbf{time}(y,\mathbf{morning})))]$$

In our example, we will assume that SNeBR is being used by a department within some university. We will follow SNeBR's behavior using the information contained in the schedules of two of its users, Stu and Tony. Both Stu and Tony already have some scheduled meetings:

1. Stu's schedule: Stu teaches a seminar in the afternoon, which is represented by the proposition **time(seminar,afternoon)**.

2. Tony's schedule: Tony has a tennis game scheduled in the morning, represented by the proposition **time(tennis-game,morning)**.

The knowledge base also contains information about which objects are meetings, i.e. **meeting(seminar)**, **meeting(tennis-game)**, **meeting(faculty-meet)**. Figure 5 shows the

knowledge base for this small example. As a shorthand, we do not represent OSs and RSs as sets of hypotheses but rather as sets of mnemonics representing the hypotheses.[23]

hyp1 : ∀(x)[meeting(x) → (time(x,morning) → ¬time(x,afternoon))] | hyp,{hyp1},{}

hyp2 : ∀(x,y)[(meeting(x) ∧ meeting(y) ∧ x≠y) →
((time(x,morning) → time(y,afternoon))
∧ (time(x,afternoon) → time(y,morning)))]
| hyp,{hyp2},{}

hyp3 : meeting(seminar) | hyp,{hyp3},{}

hyp4 : meeting(tennis-game) | hyp,{hyp4},{}

hyp5 : meeting(faculty-meet) | hyp,{hyp5},{}

hyp6 : time(seminar,afternoon) | hyp,{hyp6},{}

hyp7 : time(tennis-game,morning) | hyp,{hyp7},{}

Figure 5
Hypotheses in the knowledge base

Now suppose that Tony wants to schedule a faculty meeting and that he wants to do so according to his schedule: he considers the general statements about meetings (hyp1, hyp2, hyp4, and hyp5) and also considers the statements that reflect his schedule (hyp7). In other words, he does reasoning within the BS defined by the context Tony-schedule={hyp1, hyp2, hyp4, hyp5, hyp7}. According to our contextual interpretation, this means that the only propositions retrievable from the knowledge base by Tony are the propositions contained in the set Tony-schedule and all the propositions derivable from them. When SNeBR is asked to deduce

[23] In the SNeBR implementation, the OS of a proposition contains *links* to the hypotheses that underlie that proposition, and its RS contains *links* to the sets which are inconsistent with the proposition's OS.

when the faculty meeting will be (according to the propositions in the BS defined by the context Tony-schedule) derives the wffs in Figure 6. From this interaction, Tony concludes that the best time, for him, for scheduling the faculty meeting is in the afternoon (wff2).

wff1: time(tennis-game,morning) → time(faculty-meet,afternoon)
|der,{hyp2,hyp4,hyp5},{}

wff2: time(faculty-meet,afternoon)|der,{hyp2,hyp4,hyp5,hyp7},{}

wff3: time(faculty-meet,morning) → ¬time(faculty-meet,afternoon)
|der,{hyp1,hyp5},{}

wff4: ¬time(faculty-meet,morning)|der,{hyp1,hyp2,hyp4,hyp5,hyp7},{}

Figure 6
wffs derived from the context Tony-schedule

Suppose now that Stu also tries to find the most convenient time, for him, to have a faculty meeting. In this case, he does reasoning within the BS defined by the context Stu-schedule={hyp1, hyp2, hyp3, hyp5, hyp6}. Again, according to our contextual interpretation, the only propositions retrievable by Stu are the propositions contained in the set Stu-schedule and all the propositions derivable from them. Notice that wff3 is in this BS even though it was derived when Tony-schedule was the current context.

wff5: time(seminar,afternoon) → time(faculty-meet,morning)
|der,{hyp2,hyp3,hyp5},{}

wff6: time(faculty-meet,morning)|der,{hyp2,hyp3,hyp5,hyp6},{}

Figure 7
wffs derived from the context Stu-schedule

Figure 7 shows some of the results generated when a request is made to schedule the faculty meeting in the BS defined by the context Stu-schedule. While answering this request, SNeBR deduces that the faculty meeting is in the morning (wff 6, Figure 7). When wff6 is derived SNeBR finds out that it contradicts wff4. Since wff4 does not belong to the BS under consideration (which is now the BS defined by the context Stu-schedule) there is no visible contradiction. However, SNeBR records that the union of the contexts Stu-schedule and Tony-schedule ({hyp1, hyp2, hyp3, hyp4, hyp5, hyp6, hyp7}) is an inconsistent context. The rule URS is applied, resulting in the knowledge base represented in Figures 8 and 9. SNeBR reports to Stu that the faculty meeting should be in the morning.

hyp1: $\forall(x)[meeting(x) \rightarrow (time(x,morning) \rightarrow \neg time(x,afternoon))]$
$| hyp,\{hyp1\},\{\{hyp2,hyp3,hyp4,hyp5,hyp6,hyp7\}\}$

hyp2: $\forall(x,y)[(meeting(x) \wedge meeting(y) \wedge x \neq y) \rightarrow$
$((time(x,morning) \rightarrow time(y,afternoon)) \wedge$
$(time(x,afternoon) \rightarrow time(x,morning)))]$
$| hyp,\{hyp2\},\{\{hyp1,hyp3,hyp4,hyp5,hyp6,hyp7\}\}$

hyp3: $meeting(seminar) | hyp,\{hyp3\},\{\{hyp1,hyp2,hyp4,hyp5,hyp6,hyp7\}\}$

hyp4: $meeting(tennis\text{-}game) | hyp,\{hyp4\},\{\{hyp1,hyp2,hyp3,hyp5,hyp6,hyp7\}\}$

hyp5: $meeting(faculty\text{-}meet) | hyp,\{hyp5\},\{\{hyp1,hyp2,hyp3,hyp4,hyp6,hyp7\}\}$

hyp6: $time(seminar,morning) | hyp,\{hyp6\},\{\{hyp1,hyp2,hyp3,hyp4,hyp5,hyp7\}\}$

hyp7: $time(tennis\text{-}game,afternoon) | hyp,\{hyp7\},\{\{hyp1,hyp2,hyp3,hyp4,hyp5,hyp6\}\}$

Figure 8
Hypotheses in the knowledge base after URS

```
wff1: time(tennis-game,morning) → time(faculty-meet,afternoon)
          |der,{hyp2,hyp4,hyp5},{{hyp1,hyp3,hyp6,hyp7}}

wff2: time(faculty-meet,afternoon)
          |der,{hyp2,hyp4,hyp5,hyp7},{{hyp1,hyp3,hyp6}}

wff3: time(faculty-meet,morning) → ¬time(faculty-meet,afternoon)
          |der,{hyp1,hyp5},{{hyp2,hyp3,hyp4,hyp6,hyp7}}

wff4: ¬time(faculty-meet,morning)
          |der,{hyp1,hyp2,hyp4,hyp5,hyp7},{{hyp3,hyp6}}

wff5: time(seminar,afternoon) → time(faculty-meet,morning)
          |der,{hyp2,hyp3,hyp5},{{hyp1,hyp4,hyp6,hyp7}}

wff6: time(faculty-meet,morning)
          |der,{hyp2,hyp3,hyp5,hyp6},{{hyp1,hyp4,hyp6}}
```

Figure 9
Derived propositions in the knowledge base after URS

Suppose now that someone wants to schedule a faculty meeting with all the members of the faculty, which includes both Stu and Tony. When the request is made to consider a context containing Stu-schedule and Tony-schedule, SNeBR immediately reports that such a context is inconsistent. Notice that this context contains, possibly among others, the hypotheses hyp1, hyp2, hyp3, hyp4, hyp5, hyp6, and hyp7. The RS of hyp1, for example, is {{hyp2, hyp3, hyp4, hyp5, hyp6, hyp7}} (Figure 8), which records that the set of hypotheses hyp1 through hyp7 is inconsistent. SNeBR responds that such a context is inconsistent and that it should be revised.

Suppose that instead of Tony and Stu making the request first, a request is made to schedule the faculty meeting within a BS defined by a context containing Stu-schedule and Tony-schedule directly from the knowledge base represented in Figure 5. In this case, there are no recorded inconsistencies, and SNeBR will try to schedule the faculty meeting in that BS. Figure 10 represents some of the results derived following the query of when the faculty

meeting will be in the BS defined by a context that contains the union of the contexts Stu-schedule and Tony schedule.

wff1′: ¬**time(faculty-meet,morning)**|**der,{hyp1,hyp2,hyp4,hyp5,hyp7},{}**

wff2′: **time(faculty-meet,morning)**|**der,{hyp2,hyp3,hyp5,hyp6},{}**

Figure 10
wffs derived within a context containing both
Tony-schedule and Stu-schedule

In this case, both wff1′ and wff2′ belong to the BS under consideration (the current context contains the hypotheses hyp1, hyp2, hyp3, hyp4, hyp5, hyp6, hyp7). Therefore, when the contradiction is detected, not only is the rule of URS applied, recording the inconsistent set, but ¬I is applied also in order to rule out some hypothesis (or hypotheses)[24] defining the current context.

15. Conclusions

We presented a logic, SWM, that captures the notion of propositional dependency and is able to deal with contradictions, discussed the properties of a computer program based on SWM, and showed an example of an application of SWM.

The SWM system is loosely based on relevance logic. It associates two sets with each proposition: the *origin set* (OS) contains all the hypotheses that were used in the derivation of the proposition; the *restriction set* (RS) contains those sets of hypotheses that are incompatible with the proposition's OS.

[24] The SWM formalism guarantees that removing *exactly one hypothesis* (no matter which) produces a context which is not known to be inconsistent.

We argued that it may be desirable, in some cases, to introduce in the OS of a proposition some hypotheses that were not actually used in the derivation of that proposition. These considerations resulted from the fact that SWM is to be used as the underlying logic of a reasoning program working on a knowledge base in which one may want to conjoin propositions with different OSs. Such a conjunction cannot be allowed without restriction, due to the possible introduction of irrelevancies. The SWM system allows for this case, provided that the resulting proposition be marked as *special*. This special marking, reflected by the OT of the resulting proposition being "ext", identifies it as a proposition whose OS contains some information irrelevant to the derivation of that proposition. Every proposition in SWM whose origin tag is not "ext" effectively depends on *every* hypothesis in its OS.

SWM deals with contradictions by means of the rules of URS and $\neg$I. Upon detection of a contradiction and identification of the hypotheses contained in the OSs of the contradictory propositions as an inconsistent set, the rule of URS has the effect of recording the inconsistent set in the RS of every proposition depending on hypotheses from this set. The rule of $\neg$I allows the derivation of new propositions from a contradiction.

There are several relationships between propositions, their OS, and their RS. In fact, each proposition in SWM has a *minimal* RS, in the sense that RSs are free from some kinds of redundancies. We can also say that each proposition in SWM has a *maximal* RS in the sense that its RS records *all* inconsistent sets *known* so far. It can be shown that every proposition with the same OS has the same RS, reflecting the fact that RSs are both minimal and maximal.

In MBR, an abstract belief revision system based on SWM, the notions of *context* and *belief space* are defined. A context is any set of hypotheses. A context determines a belief space (BS), which is the set of all propositions whose OS is contained in the context. In other words, a BS contains all the propositions that depend exclusively on the hypotheses defining the context. The notions of context and BS provide a contextual interpretation for the propositions in a SWM based system. In fact, given any context, the only propositions whose truth

value is *known* are those propositions that belong to the BS defined by the context. The truth value of all the other propositions is unknown. By a proposition having an unknown truth value, we mean that in order to compute its truth value one has to carry out further deduction, and it may even be possible that its truth value is not computable from the hypotheses under consideration.

Queries to MBR are associated with a context. Following a given query, the only propositions that are retrievable from the knowledge base are the ones that belong to the current BS (the BS defined by the context in the query).

We discussed the cases in which the rules of URS and ¬I should be applied: When a contradiction is found and one of the contradictory propositions does not belong to the current BS, then URS must be applied, recording the contradiction and ¬I is not applicable; if both contradictory propositions belong to the current BS, then URS must be applied and ¬I should be applied if one decides to get rid of the contradiction.

When a contradiction is detected in the current BS and — after selecting one or more hypotheses as the culprits for the contradiction — one is faced with the problem of making inaccessible to the automatic reasoning system all the propositions that were previously derived from such hypotheses, all one has to do in MBR is remove the selected hypotheses from the context in the following queries. Afterwards, all the propositions derived from the selected hypotheses are no longer in the current BS (since their OS is no longer contained in the current context) and consequently are not retrievable by the deduction system.

The same mechanism may be used to delineate the beliefs of different individuals. Letting the current context be the set of hypotheses believed by some individual has the effect of limiting the attention of the inference system to the BS of that individual and thus separating his/her beliefs from the beliefs of all the other individuals, in the sense that only the propositions that he/she believes are available to the deduction system.

Finally, SWM and MBR have been fully implemented in a computer system, SNeBR, written in Franz Lisp and using SNePS.

16. Acknowledgements

Many thanks to John Corcoran, Jon Doyle, Donald McKay, Ernesto Morgado, Jane Terry Nutter, William J. Rapaport, and members of the SNePS Research Group for their criticisms and suggestions.

This work was partially supported by the National Science Foundation under Grant MCS80-06314 and by the Instituto Nacional de Invertigação Cientifica, under Grant no.20536. Preparation of this paper was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008.

17. References

Anderson A. and Belnap N., **Entailment: The Logic of Relevance and Necessity** Vol.1, Princeton University Press, 1975.

Charniak E., Riesbeck C. and McDermott D., **Artificial Intelligence Programming**, Lawrence Erlbaum Associates, 1980.

de Kleer J., "Choices without Backtracking", **Proc. Conference of the American Association for Artificial Intelligence 1984**, pp. 79-85, William Kaufmann.

Doyle J., "Truth Maintenance Systems for Problem Solving", Technical Report AI-TR-419, Massachussets Institute of Technology, AI Lab., 1978

Doyle J., "A Truth Maintenance System", **Artificial Intelligence**, Vol.12 No.3, pp.231-272, 1979.

Doyle J., "A model for Deliberation, Action and Introspection", Ph.D. Dissertation, Technical Report No.581, Massachussets Institute of Technology, AI Lab, 1980.

Doyle J., "Some Theories of Reasoned Assumptions: An Essay in Rational Psychology", Department of Computer Science, Carnegie-Mellon University, 1982.

Doyle J., "The Ins and Outs of Reason Maintenance", **Proc. International Joint Conference on Artificial Intelligence 1983**, pp.349-351, William Kaufmann.

Doyle J. and London P., "A selected descriptor-indexed bibliography to the literature of Belief Revision", **SIGART Newsletter 71**, pp.7-23, 1980.

Fahlman S., "A Planning System for Robot Construction Tasks", **Artificial Intelligence**, Vol.5, no.1, pp.1-49, 1974.

Fikes R., "Deductive Retrieval Mechanisms for State Description Models", **Proc. International Joint Conference on Artificial Intelligence 1975**, pp.99-106, William Kaufmann.

Fikes R. and Nilson N., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", **Artificial Intelligence**, Vol.2, No.3-4, pp.189-208, 1971.

Fitch F., **Symbolic Logic: An Introduction**, Ronald Press, 1952.

Foderaro J., Sklower K. and Layer K., "The Franz Lisp Manual", UNIX Programmers Manual - Supplementary Documents, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1984.

Goodwin J. W., "A Improved Algorithm for Non-monotonic Dependency Net Update", Research Report LiTH-MAT-R-82-23, Software Systems Research Center, Linkoping Institute of Technology, Sweden, August 1982.

Goodwin J.W., "WATSON: A Dependency Directed Inference System", **Proc. Non-Monotonic Reasoning Workshop**, pp.103-114, American Association for Artificial Intelligence, 1984.

Hayes P.J., "The Frame Problem and Related Problems in Artificial Intelligence", in **Artificial and Human Thinking**, Elithorn and Jones (eds.), pp.45-59, Jossey-Bass Inc., 1973.

Hewitt C., "Description and Theoretical Analysis of PLANNER: A Language for Proving -Theorems and Manipulating Models in a Robot", Technical Report TR-258, Massachussets Institute of Technology, 1972.

Konolige K., "Belief and Incompleteness", in **Formal Theories of the Commonsense World**, J. Hobbs and R. Moore (eds.),pp.359-403, Ablex Publishing, 1985.

London P., "Dependency Networks as Representation for Modelling in General Problem Solvers", Ph.D. Dissertation, Technical Report 698, Department of Computer Science, University of Maryland, 1978.

Martins J., "Reasoning in Multiple Belief Spaces", Ph.D. Dissertation, Technical Report 203, Department of Computer Science, State University of New York at Buffalo, 1983.

Martins J., "Belief Revision", in **Encyclopedia of Artificial Intelligence**, S.C. Shapiro (ed.), John Wiley and Sons, forthcoming.

Martins J. and Shapiro S., "Reasoning in Multiple Belief Spaces", **Proc. International Joint**

Conference on Artificial Intelligence 1983, pp.370-373, William Kaufmann.

Martins J. and Shapiro S., "A Model for Belief Revision", forthcoming.

McAllester D., "An Outlook on Truth-Maintenance", AI Memo 551, Massachussets Institute of Technology, AI Lab., 1980.

McCarthy J. and Hayes P., "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in **Machine Intelligence 4**, Meltzer and Michie (eds.), pp.463-502, Edinburgh University Press, 1969.

McDermott D., "Contexts and Data Dependencies: A Synthesis", Department of Computer Science, Yale University, 1982.

McDermott D., "Data Dependencies on Inequalities", **Proc. Conference of the American Association for Artificial Intelligence 1983**, pp.266-269, William Kaufmann.

McDermott D. and Sussman G., "The CONNIVER Reference Manual", Technical Report Memo 259, Massachussets Institute of Technology, 1972.

McKay D and Martins J., "Provisional SNePSLOG User's Manual" Department of Computer Science, State University of New York at Buffalo, 1981.

Raphael B., "The Frame Problem in Problem Solving Systems", in **Artificial Intelligence and Heuristic Programming**, Findler and Meltzer (eds.), pp.159-169, American Elsevier, 1971.

Rips L.J., "Cognitive Processes in Propositional Reasoning", **Psychological Review**, Vol.90, No.1, pp.38-71, 1983.

Rulifson J., Derksen J. and Waldinger R., "QA4: A Procedural Calculus for Intuitive Reasoning", Technical Report Note 73, SRI International, 1972.

Shapiro S., "The SNePS Semantic Network Processing System", in **Associative Networks**, Findler (ed.), pp.179-203, Academic Press, 1979.

Shapiro S. and Wand M., "The Relevance of Relevance", Technical Report No. 46, Computer Science Department, Indiana University, Bloomington, Indiana, 1976.

Shrobe E., "Dependency-directed Reasoning in the Analysis of Programs which Modify Complex Data Structures", **Proc. International Joint Conference on Artificial Intelligence 1979**, pp.829-835, William Kaufmann.

Stallman R. and Sussman G., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", **Artificial Intelligence**, Vol.9, No.2, pp.135-196, 1977.

Sussman G., Winograd T. and Charniak E., "MICRO-PLANNER Reference Manual", Technical Report Memo 203, Massachussets Institute of Technology, 1971.

Tarski A., **An Introduction to Logic and to the Methodology of Deductive Sciences**, Oxford University Press, 1965 (3rd. edition).

Thompson A., "Network Truth-Maintenance for Deduction and Modelling", **Proc. International Joint Conference on Artificial Intelligence 1979**, pp.877-879, William Kaufmann.

Appendix 1: Proof of theorems

In this appendix, we present the proofs of the theorems mentioned in the text.

Theorem 1: For every set A and B $\sigma(\Psi(A,B)) \subset \Psi(\sigma(A),B)$.

Proof: We have to show that

$$(\forall x)[x \in \sigma(\Psi(A,B)) \rightarrow x \in \Psi(\sigma(A),B)].$$

Suppose by way of contradiction that $\exists x \in \sigma(\Psi(A,B))$ such that $x \notin \Psi(\sigma(A),B)$. Since $x \in \sigma(\Psi(A,B))$ we know that

1. $\forall \alpha \in \Psi(A,B)$, $\alpha \neq x$ then $\alpha \subset x$

2. $x \cap B = \emptyset$

and that one of the following cases holds:

3. $x \in A$

3' $\exists \beta \subset B : (x \cup \beta) \in A$ $(\beta \neq \emptyset)$.

If case 3 holds we can conclude that

4. $x \in \sigma(A)$

which, together with 2, implies that $x \in \Psi(\sigma(A),B)$, which is a contradiction.

If case 3' holds, we know that

4'. $(x \cup \beta) \notin A$

From $x \notin \Psi(\sigma(A),B)$ we can infer that

5'. $(x \cup \beta) \notin \sigma(A)$

from 4' and 5', it follows that

6' $\exists \gamma \in A : \gamma \subset (x \cup \beta)$ and $\gamma \notin \sigma(A)$.

Letting $\gamma' = \gamma - B$ it is clear that $\gamma' \in \Psi(A,B)$. From 2 and 6' it follows that $\gamma' \subset x$, which contradicts 1. □

Theorem 2: Given the supported wffs A | t_a,o_a,r_a and B | t_b,o_b,r_b then the supported wff C | $t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ has a *minimal* RS, regardless of whether or not A and B have minimal RSs.

Proof: Suppose by way of contradiction that given A | t_a,o_a,r_a and B | t_b,o_b,r_b, the supported wff C | $t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ does not have a minimal RS. This entails that one of the following two cases holds:

1. $\exists r \in \mu(\{r_a,r_b\},\{o_a,o_b\})$: $r \cap (o_a \cup o_b) \neq \emptyset$

2. $\exists r,s \in \mu(\{r_a,r_b\},\{o_a,o_b\})$: $r \subset s$.

Suppose that condition 1 holds. Since $r \in \mu(\{r_a,r_b\},\{o_a,o_b\})$ we can infer that $r \in \psi(r_a \cup r_b, o_a \cup o_b)$, which, due to the way ψ was defined, contradicts the fact that $r \cap (o_a \cup o_b) \neq \emptyset$.

Suppose that condition 2 holds. Since $r,s \in \mu(\{r_a,r_b\},\{o_a,o_b\})$ we can infer that $r,s \in \sigma(\psi(r_a \cup r_b, o_a \cup o_b))$, which, due to the way σ was defined, contradicts the fact that $r \subset s$.

Therefore C | $t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ has minimal RS. □

Theorem 3: Given the supported wffs A | $t_a,o_a,\{R_1, \ldots, R_n\}$ and B | t_b,o_b,r_b then C | $t_c,o_a - o_b,(o_a - o_b)$ has minimal RS.

Proof: Suppose that $o_a - o_b = \{H_1, \ldots, H_n\}$. Taking into account that C | $t_c,o_a - o_b,(o_a - o_b)$ is the same as C | $t_c,o_a - o_b,\mu(\mathbf{rs}(H_1), \ldots, \mathbf{rs}(H_n)),\{\{H_1\}, \ldots, \{H_n\}\})$ the statement of this theorem follows from Theorem 2. □

Theorem 4: If A is an inconsistent set then so is any set containing A.

Proof: The proof presented is based on the fact that a proof of B from $\{A_1, \ldots, A_n\}$ is a sequence of lines, the first n of which are $A_1, \ldots, A_n$ and the last of which is

B. Each line between A_n and B is obtained from the previous line(s) by the application of some rule of inference and is justified by $R(L_1,L_2)$ in which R represents some rule of inference and (L_1,L_2) are the line numbers of the parent wffs.

Suppose that $A=\{P_1, \ldots, P_n\}$ and let $r_1(L_{11},L_{12})$, $r_2(L_{21},L_{22})$, $\ldots$, $r_k(L_{k1},L_{k2})$ represent the sequence of applications of rules of inference to the (ordered) set A (and to the wffs derived from them) to generate a $\rightarrow\leftarrow$. Suppose that $A \subset B$ and $B-A=\{S_1, \ldots, S_i\}$. B can, therefore, be written as the following ordered set (in which the order of $S_1, \ldots, S_i$ is irrelevant) $B=\{S_1, \ldots, S_i, P_1, \ldots, P_n\}$. Then, letting $L'_{fg}=L_{fg}+i$, the following sequence of rules of inference $r_1(L'_{11},L'_{12})$, $r_2(L'_{21},L'_{22})$, $\ldots$, $r_k(L'_{k1},L'_{k2})$ describes a derivation of a $\rightarrow\leftarrow$ from B. □

Corollary 4.1: If A is not known to be inconsistent then neither is any set contained in A.

Proof: Suppose by way of contradiction that A is not known to be inconsistent and that $B \subset A$ is known to be inconsistent. By the Theorem 4, A is known to be inconsistent, which is a contradiction. □

Theorem 5: All the supported wffs in the knowledge base resulting from the application of the rules of inference of the SWM system have minimal RSs.

Proof: The proof will be done by complete induction on the number of applications of rules of inference.

The only supported wffs which can be obtained by applying one rule of inference only are hypotheses and the rule of Hvp guarantees that they have minimal RSs.

Suppose now, by induction hypothesis, that all the supported wffs obtained by the application of less than *n* rules of inference have minimal RSs. We will have to prove that the supported wff obtained by the application of *n* rules of inference has

minimal RSs as well.

1. The supported wff is obtained by the application of either $\neg E$, $\wedge E$, $\forall I$, $\vee I$, $\exists I$, or $\exists E$ then it has the same OS and RS as the parent supported wff and consequently has minimal RS.

2. The supported wff is obtained by the application of either MP, MT, $\wedge I$, $\vee E$, or $\forall E$, then by Theorem 2 we may conclude that it has minimal RS.

3. The supported wff is obtained by the application of $\rightarrow I$, or $\forall I$ then by Theorem 3 we can conclude that it has minimal RS.

4. The supported wff is obtained by $\neg I$. We will show that a supported wff obtained using the second part of the rule of $\neg I$ has minimal RS (to show that a supported wff obtained using the first part of the rule of $\neg I$ has minimal RS as well it suffices to take $o_1=o_2=\emptyset$ in the proof that follows). Consider $A \mid t_1,o_1,r_1$ and $\neg A \mid t_2,o_2,r_2$. For $\{H_1, \ldots, H_n\} \subset (o_1 \cup o_2)$, the second part of the rule of $\neg I$ allows us to deduce $\neg(H_1 \wedge \ldots \wedge H_n) \mid$ ext, $o_1 \cup o_2, (o_a \cup o_b - \{H_1, \ldots, H_n\})$. To show that this supported wff has minimal RS let us suppose that it is obtained in two steps: first, deduce $A \wedge \neg A \mid$ ext, $o_1 \cup o_2, \mu(\{r_1,r_2\},\{o_1,o_2\})$; second, from the above supported wff deduce $\neg(H_1 \wedge \ldots \wedge H_n) \mid$ ext, $o_1 \cup o_2 - \{H_1, \ldots, H_n\}, (o_a \cup o_b - \{H_1, \ldots, H_n\})$. by Theorem 3 this supported wff has minimal RS.

5. The supported wff is obtained by URS. We will consider two sub-cases:

a. The supported wff results from updating the RS of an hypothesis. Suppose that the supported wff is obtained by updating $H \mid$ hyp,$\{H\}$,R. We know that prior to updating H has minimal RS. We want to show that $H \mid$ hyp,$\{H\}$,$\sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ verifies the following conditions:

1. $\forall r \in \sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ $r \cap \{H\} = \emptyset$

2. $\forall r,s \in \sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ $r \not\subset s$.

Condition 1 holds since $\forall r \in R$, $r \cap \{H\} = \emptyset$ (before the update H had minimal RS) and $[(o_1 \cup o_2) - \{H\}] \cap \{H\} = \emptyset$ (by definition). Condition 2 is verified by definition of σ.

b. The supported wff results from updating the RS of a supported wff whose OT is either "der" or "ext". Suppose that the supported wff is obtained by updating F | t,o,R ($t \neq hyp$). We want to show that F | t,o,$\sigma(R \cup \{(o_1 \cup o_2)-o\})$ verifies the following two conditions:

1. $\forall r \in \sigma(R \cup \{(o_1 \cup o_2)-o\})$ $r \cap o = \emptyset$

2. $\forall r,s \in \sigma(R \cup \{(o_1 \cup o_2)-o\})$ $r \not\subset s$.

Suppose by way of contradiction that condition 1 does not hold. Since F | t,o,R has minimal RS we know that $\forall r \in R$, $r \cap o = \emptyset$, thereby $[(o_1 \cup o_2)-o] \cap o \neq \emptyset$, which is a contradiction, therefore condition 1 holds. Condition 2 follows from the definition of σ.

Therefore all the supported wffs in the knowledge base resulting from the application of the rules of inference of the SWM system have minimal RS. □

Lemma 1: Given n supported wffs $F_1 \mid t_1,o_1,r_1, \ldots, F_n \mid t_n,o_n,r_n$ then the sets $R_1=\mu(\{r_1, \ldots ,r_n\},\{o_1, \ldots ,o_n\})$ and $R_2=\mu(\mu(\{r_1, \ldots, r_i\},\{o_1, \ldots ,o_i\}), \mu(\{r_{i+1}, \ldots ,r_n\},\{o_{i+1}, \ldots, o_n\}),$ $\{o_1, \ldots ,o_n\})$ $(1 \leq i \leq n-1)$ are equal.

Proof: Suppose by way of contradiction that $R_1 \neq R_2$. This means that either $\exists r \in R_1$ such that $r \notin R_2$ or that $\exists r \in R_2$ such that $r \notin R_1$. We will consider each of these cases in turn:

1. Suppose that $r \in R_1$ and that $r \notin R_2$. Suppose furthermore that r was originated from a set s, i.e., $r=s-(o_1 \cup \ldots \cup o_n)$ and suppose that $s \in r_j$. Letting $O=o_1 \cup \ldots \cup o_n$ we have that $r=s-O$. The fact that r belongs to R_1 means that $\forall r_k \in \{r_1, \ldots ,r_n\}$ and $r_k \neq r_j$ $\neg(\exists u \in r_k : u-O \subset s-O)$. Since $r \notin R_2$, the set s was deleted either by one of the following applications of the operation μ, $\mu(\{r_1, \ldots ,r_i\},\{o_1, \ldots ,r_i\})=R'$ or $\mu(\{r_{i+1}, \ldots ,r_n\},\{o_{i+1}, \ldots ,o_n\})=R''$, or else s was deleted by $\mu(\{R',R''\},\{o_1, \ldots ,o_n\})$.

a. Suppose that s was deleted while creating the set R' (if s was deleted while creating the set R" the reasoning would be similar) this means that $r_j \in \{r_1, \ldots ,r_i\}$. It also means that

$\exists r_q \in \{r_1, \ldots, r_i\}$ such that $\exists p \in r_q : p-(o_1 \cup \ldots \cup o_i) \subset s-(o_1 \cup \ldots \cup o_i)$. Therefore $p-O \subset s-O$ which is a contradiction.

b. Suppose that s was deleted while creating R_2, i.e., either $s-(o_1 \cup \ldots \cup o_i) \notin R'$ or $s-(o_{i+1} \cup \ldots \cup o_n) \notin R''$. In either case, it means that $\exists r_p \in \{r_1, \ldots, r_n\}$ $r_p \neq r_j$ and $\exists u \in r_p : (u-O \subset s-O)$ which is a contradiction.

2. Suppose that $r \in R_2$ and that $r \notin R_1$. Suppose furthermore that r was originated by a set $s \in r_j$, i.e., $r=s-O$. Since $r \notin R_1$ this means that $\exists r_k \in \{r_1, \ldots, r_n\}$, $r_k \neq r_j$ and $\exists u \in r_k : (u-O \subset s-O)$. Since $r \in R_2$, it means that s was not deleted while creating the sets R' and R", nor was it deleted while creating R_2. We will examine the consequences of each of these two cases:

a. Suppose that both r_j and r_k belong to one of $\{r_1, \ldots, r_i\}$ or $\{r_{i+1}, \ldots, r_n\}$, say that they both belong to $\{r_1, \ldots, r_i\}$. Then, since s was not deleted while creating R' it means that $[u-(o_1 \cup \ldots \cup o_i)] \subset [s-(o_1 \cup \ldots \cup o_i)]$. Now, if u is not deleted by the application of μ which creates R', then both u-O and s-O will be considered while creating R_2 and s-O will be deleted by the application of μ, which is a contradiction; if u is deleted while creating R' then $\exists r_q \in \{r_1, \ldots, r_i\}$ such that $\exists p \in r_q : p-(o_1 \cup \ldots \cup o_i) \subset u-(o_1 \cup \ldots \cup o_i)$, meaning that both $p-(o_1 \cup \ldots \cup o_i)$ and $s-(o_1 \cup \ldots \cup o_i)$ belong to R' and therefore s will be deleted by the application of μ which creates R_2, which is a contradiction.

b. Suppose that r_j and r_k do not both belong to one of $\{r_1, \ldots, r_i\}$ and $\{r_{i+1}, \ldots, r_n\}$. Then s would be deleted by the application of μ which creates R_2, which is a contradiction.

Therefore $R_1=R_2$. □

Lemma 2: Given two supported wffs with the same OS, if their RS was obtained exclusively by successive applications of the μ operation then the supported wffs have the same RS as well.

Proof: It follows directly from Lemma 1. □

Theorem 6: In the knowledge base resulting from the application of the rules of inference of the SWM system, if two supported wffs have the same OS then they have the same RS as well.

Proof: The proof will be done by complete induction on the number of applications of rules of inference.

The only supported wffs which can be obtained by applying one rule of inference only are hypotheses and since the rule of hyp guarantees that their OS is unique the theorem is verified.

Suppose, by induction hypothesis, that all the supported wffs obtained by the application of less than n rules of inference verify the conditions of the theorem. We will have to show that the supported wff obtained by the application of the n-th rule of inference also satisfies the statement of the theorem. We will group the rules of inference of SWM according to the type of OS and RS they produce and will discuss the OS and RS of the supported wff produced by the application of such type of rule.

1. Creation of new OSs (hyp). The rule of hyp creates a supported wff with a new OS. The assumption behind the application of this rule is that there is no supported wff in the knowledge base with such OS and therefore this supported wff satisfies the conditions of the theorem.

2. Change in RS only (URS). Upon detection of an inconsistent set, χ, this rule changes the RS of every supported wff whose OS is not disjoint from χ. Every supported wff $F \mid t,o,R$ such that $\chi \cap o \neq \emptyset$ is replaced by $F \mid t,o,\sigma(R \cup \{\chi - o\})$. If prior to the application of this rule all the supported wffs with the same OS had the same RS the same condition will

be verified after the application of the rule since the RS's of supported wffs with the same OS are affected in the same way. An important point to note is that $\sigma(R \cup \{\chi - o\}) = \mu(R \cup \{\chi - o\}, o)$ and thus the RS of the resulting supported wffs are the same as if the μ operation had been applied.

3. No change in OS nor RS ($\neg$E, $\wedge$I(part 1), $\wedge$E, ∇I, $\vee$I, $\exists$I, $\exists$E). The application of one of these rules creates a new supported wff whose OS and RS are the same as the OS and RS of a previous supported wff, therefore, by the induction hypotheses the statement of the theorem is verified.

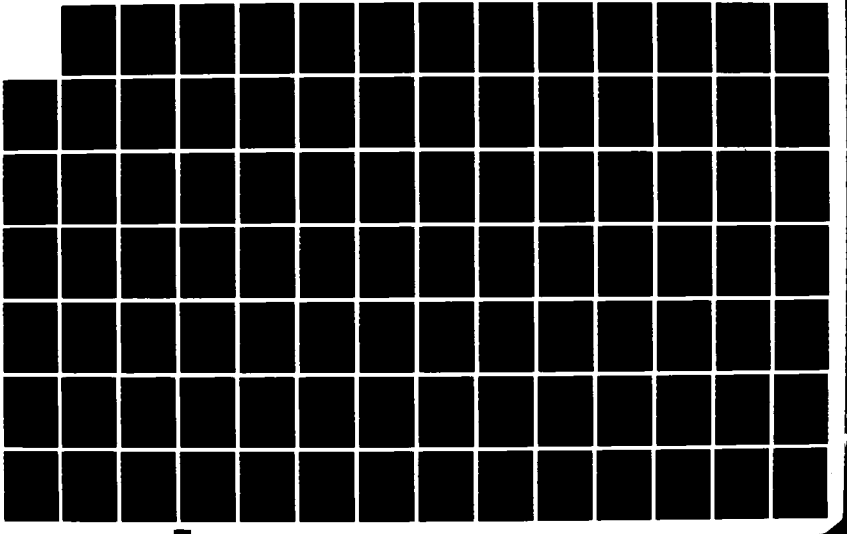
4. The OS is the union of the OSs of the parent wffs (MP, MT, $\wedge$I(part 2), $\vee$E, $\forall$E). Using one of these rules, if we combine $F_1 \mid t_1, o_1, r_1$ and $F_2 \mid t_2, o_2, r_2$ we obtain $F_3 \mid t_3, o_1 \cup o_2, \mu(\{r_1, r_2\}, \{o_1, o_2\})$. Lemma 2 guarantees that, if only μ operations had been applied to form the RSs of the supported wffs in the knowledge base, the RS of F_3 is the same as the RS of any wff whose OS is $o_1 \cup o_2$.

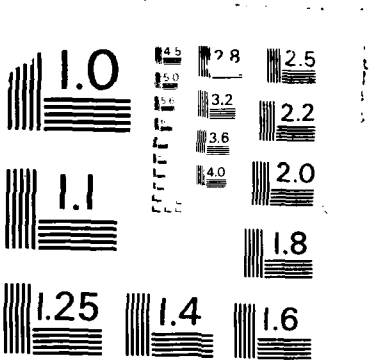
5. The resulting OS is one (or several) hypothesis short ($\rightarrow$I, $\neg$I, $\forall$I). Using one of these rules, we take one supported wff $F_1 \mid t_1, o, r$ and create a new supported wff $F_2 \mid t_2, o - \{H_1, \ldots, H_n\}, (o - \{H_1, \ldots, H_n\})$, recall that $(O) = \mu(\{r : \exists h \in O : r = \mathbf{rs}(h)\}, \{o : \exists h \in O : o = \mathbf{os}(h)\})$ therefore Lemma 2 guarantees that, if only μ operations had been applied to form the RSs of the supported wffs in the knowledge base, the statement of the theorem is verified.

As a final remark it should be noticed that since the application of [illegible] an application of μ, the RS of *every* supported wff in the knowledge base [illegible] sive applications of the μ operation and therefore if two supported wffs have the same OS they also have the same RS.

Corollary 6.1 [illegible]

AD-A189 773 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC) 2/5
REVIEW OF TECHNICAL T.. (U) NORTHEAST ARTIFICIAL
INTELLIGENCE CONSORTIUM SYRACUSE NY J F ALLEN ET AL.
UNCLASSIFIED JUL 87 RADC-TR-86-218-V2-PT1 F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963

Theorem 7: Suppose that $C=\{H_1, \ldots, H_n\}$ is a context which is not known to be inconsistent. Then, for any two wffs, say A and B, in the BS defined by the context C, we have Combine(A,B)=true.

Proof: Suppose by way of contradiction that the wffs $A \mid t_a,o_a,r_a$ and $B \mid t_b,o_b,r_b$ belong to the BS defined by the context C and that Combine(A,B)=false. Since A and B belong to the BS defined by C we have $o_a \subset C$ and $o_b \subset C$. Since Combine(A,B)=false, one of the following conditions holds:

1. $\exists\ r \in \mathbf{rs}(A) : r \subset \mathbf{os}(B)$: since $\mathbf{os}(B) \subset \{H_1, \ldots, H_n\}$ we have that $r \subset \{H_1, \ldots, H_n\}$. By definition of restriction set we know that $(o_a \cup r) \vdash \rightarrow\leftarrow$. Since $o_a \subset \{H_1, \ldots, H_n\}$ and $r \subset \{H_1, \ldots, H_n\}$ we have that $(o_a \cup r) \subset \{H_1, \ldots, H_n\}$. Therefore, there exists a set $S \subset \{H_1, \ldots, H_n\}$ such that $S \vdash \rightarrow\leftarrow$. By Theorem 4 $\{H_1, \ldots, H_n\} \vdash \rightarrow\leftarrow$ which contradicts the assumption that C is not known to be inconsistent.

2. $\exists\ r \in \mathbf{rs}(B) : r \subset \mathbf{os}(A)$: the same line of reasoning used in 1 will derive a contradiction.

Therefore Combine(A,B)=true. □

Corollary 7.1: If one uses a context which is not known to be inconsistent then the system using SWM does not need to check for combinability between the wffs before the application of rules of inference.

A Model for Belief Revision

by

João P. Martins [*] and Stuart C. Shapiro [**]

[*] Departamento de Engenharia Mecanica
Instituto Superior Tecnico
Av. Rovisco Pais
1000 Lisboa, Portugal

[**] Department of Computer Science
University at Buffalo
State University of New York
226 Bell Hall
Buffalo, New York 14260, U.S.A.

Abstract

It is generally recognized that the possibility of detecting contradictions and identifying their sources is an important feature of an intelligent system. Systems that are able to detect contradictions and/or identify their causes, called Belief Revision Systems, Truth Maintenance Systems, or Reason Maintenance Systems, have been implemented by several researchers in Artificial Intelligence (AI).

In this paper, we present a logic suitable to support belief revision systems, discuss the properties that a belief revision system based on this logic will exhibit, and present a particular implementation of our model of a belief revision system. The system we present, SNeBR, differs from most of the systems developed so far in two respects: First, it is based on a logic which was developed to support belief revision systems. Second, its implementation relies on the manipulation of sets of assumptions, not justifications. The first feature allows the study of the formal properties of the system independently of its implementation, and the second one enables the system to work effectively and efficiently with inconsistent information, to switch reasoning contexts without processing overhead, and to avoid most backtracking.

1. Introduction

Belief revision systems [Doyle and London 80; Martins forthcoming] are AI programs that deal with contradictions. They work with a knowledge base, performing reasoning from the propositions in the knowledge base and "filtering" the propositions in the knowledge base so that only part of the knowledge base is perceived — the set of propositions that are under consideration. This set of propositions is usually called the set of *believed propositions*. When the belief revision system considers another of these sets, it is usual to say that it *changes its beliefs*. Typically, the belief revision system explores alternatives, makes choices, explores the consequences of its choices, and compares results obtained when using different choices. If during this process a contradiction is detected, then the belief revision system will revise the knowledge base, "erasing" some propositions so that it gets rid of the contradiction.

In order to illustrate some of the issues in belief revision, we will consider the following puzzle from [Summers 72]:

> Freeman knows five women: Ada, Bea, Cyd, Deb and Eve. The women are in two age brackets: three women are under 30 and two women are over 30. Two women are teachers and the other three women are secretaries. Ada and Cyd are in the same age bracket. Deb and Eve are in different age brackets. Bea and Eve have the same occupation. Cyd and Deb have different occupations. Of the five women, Freeman will marry the teacher over 30. Who will Freeman marry?

Suppose that we wanted to solve this puzzle. One way of doing this is to raise some hypotheses concerning the women's ages and professions and reason from the hypotheses raised. If we obtain a contradiction then we change some of these hypotheses and proceed.

Let us assume that Ada and Cyd are over 30, and that Ada and Deb are teachers. Clearly, in this case, Freeman will marry Ada. It remains to verify whether the hypotheses raised are consistent with the puzzle's statement: Since there are only two women over 30, it follows that Bea, Deb, and Eve are under 30, which is a contradiction since Deb and Eve are in the same age bracket. Thus, we conclude that the hypotheses raised about the woman's ages are not consistent with the puzzle's statement and have to be changed.

Let us now assume that Bea and Deb are the women over 30, meaning that Freeman marries Deb (remember that we did not change the hypotheses concerning the women's professions). In this case, Ada, Cyd and Eve are under 30, Ada and Cyd are in the same age bracket, and Deb and Eve are in different age brackets, and thus the ages hypothesized are consistent with the puzzle's statement. Now, in respect to the professions of the women, Bea, Cyd and Eve are secretaries, Bea and Eve having the same occupation, and Cyd and Deb having different occupations which is also consistent with the statement of the puzzle. Thus Freeman will marry Deb.

This example shows the type of reasoning one wants to perform with a belief revision system: raise hypothesis, reason from them, and if a contradiction is detected identify which hypotheses may have contributed to the contradiction, change some of them and resume the reasoning. In this paper we present a belief revision system that is able to do this, and furthermore allows sentences like "two woman are teachers", "three women are secretaries" to be expressed in a very simple way. In Section 8 we present a run in which our system solves this puzzle.

In this paper, we present a logic suitable to support belief revision systems, discuss the properties that a belief revision system based on this logic will exhibit, and present a particular implementation of our model of a belief revision system. The system we present, SNeBR, differs from most of the systems developed so far in two respects: First, it is based on a logic which was developed to support belief revision systems. Second, its implementation relies on the manipulation of sets of assumptions, not justifications. The first feature allows the study of the formal properties of the system independently of its implementation, and the second one enables the system to work effectively and efficiently with inconsistent information, to switch reasoning contexts without processing overhead, and to avoid most backtracking.

In this paper, we address the following issues: What are the alternative methods of recording dependencies of propositions in belief revision systems? What are the advantages

and disadvantages of each of these? What kind of logic should underlie a computer program used for applications in belief revision? What techniques should be supplied by the logic for coping with the possible occurrence of contradictions? How should a computer program based on that logic interpret the techniques in order to recover from contradictions and to avoid reasoning from contradictory hypotheses?

The paper is organized into several sections: Section 2 discuss the general background of work in AI related to belief revision. Section 3 presents a discussion about two alternatives to represent dependencies of propositions. Section 4 presents a logic, the SWM system, that relies on the notion of dependency and provides for dealing with contradictions. SWM is loosely based on relevance logic it associates each proposition with the set of hypothesis that were *really* used in its derivation and with all the sets of hypothesis with which it is incompatible. Section 5 SWM by the introduction of non-standard connectives. Section 6 discusses the features that a computer system based on SWM will exhibit. In this section, an abstract model of a belief revision system (i.e., not tied to any particular implementation) is defined. In this system, MBR, the notions of context and belief space are defined. A *context* is any set of hypotheses. A context determines a *belief space*, which is the set of all propositions depending exclusively on the hypotheses defining the context. The only propositions retrievable from the knowledge base at any moment are the propositions that belong to the belief space being considered. Section 7 presents a particular implementation of our abstract model using the SNePS semantic network processing system. The resulting system, SNeBR, is written in Franz Lisp and runs on VAX-11 systems. Section 8 shows an example using SNeBR.

2. Background

The ability to reason about and adapt to a changing environment is an important aspect of intelligent behavior. Most computer programs constructed by researchers in AI maintain a model of their environment (external and or internal), which is updated to reflect the perceived changes in the environment. The model of the environment is typically stored in a

knowledge-base (containing propositions about the state of the environment) and the program manipulates the information in this knowledge base. Most of the manipulation consists of drawing inferences from information in the knowledge base. All the inferences drawn are added to the knowledge base. One reason for model updating (and thus knowledge base updating) is the detection of contradictory information about the environment. In this case the updating should be preceded by the decision of what proposition in the knowledge base is the culprit for the contradiction, its removal from the knowledge base[1], and the subsequent removal from the knowledge base of every proposition that depends on the selected culprit.

The conventional approach to handling *contradictions* consists of changing the most recent decision made, i.e., the contradiction is blamed on the most recent decision made (chronological backtracking). An alternative solution (dependency-directed backtracking) consists of changing, not the last choice made, but the choice that *most likely* caused the unexpected condition to occur. This second approach, proposed in the late 70's by Stallman and Sussman [Stallman and Sussman 77], originated a great deal of research in one area of AI which became loosely called Belief Revision.

Belief revision systems [Doyle and London 80; Martins forthcoming] are AI programs that deal with contradictions. They work with a knowledge base, performing reasoning from the propositions in the knowledge base and "filtering" the propositions in the knowledge base so that only part of the knowledge base is perceived — the set of propositions that are under consideration. This set of propositions is usually called the set of *believed propositions*. When the belief revision system decides to consider another of these sets, it is usual to say that it *changes its beliefs*. Typically, belief revision systems explore alternatives, make choices, explore the consequences of its choices, and compare results obtained when using different choices. If during this process a contradiction is detected (i.e., both a proposition and its nega-

[1]Or making it inaccessible to the program.

tion belong to the set of believed propositions), then the belief revision system will revise the knowledge base, "erasing" some propositions so that it gets rid of the contradiction.[2]

Belief revision systems in AI have their roots in the frame problem [Hayes 73, McCarthy and Hayes 69, Raphael 71], the problem of deciding which conditions change and which conditions do not change in a system when it undergoes some modification. The basis of the problem is that although it is possible to specify the ways in which a system's environment might change in terms of effects of actions, it still remains to specify some way of deciding what stays unchanged in face of the actions.

An important advance was made by Stallman and Sussman, who designed a system, called EL, in which dependencies of propositions are permanently recorded [Stallman and Sussman 77]. EL maintains a complete record (trace) of its reasoning, using it both to decide which alternative choices to make when something goes wrong and to explain its line of reasoning. Along with each derived proposition, EL stores the set of all propositions directly used in its derivation and the rule of inference used to derive it; this is the *dependency record* of the proposition. EL solves electric circuit problems. While searching for the values of the circuit parameters, EL may have to "guess" the operating range of some devices. Later on, if an inconsistency is found, EL knows that somewhere along its way it guessed a wrong state for some device. The novelty of EL's approach to backtracking is that the assumption that is changed during backtracking does not necessarily correspond to the last choice made but rather to the assumption that *most likely* caused the contradiction to occur (*dependency-directed backtracking*). When an inconsistency is detected, EL searches through the chain of dependency records of the inconsistent propositions until it finds all the assumptions (guesses made) upon which the inconsistent propositions depend. Then heuristics are used to rule out one of them. This set of assumptions is recorded as leading to a contradiction and is never tried again. Stallman

[2]There are some cases in which it is desirable to continue reasoning within a knowledge base in which a contradiction exists; for a description of this type of reasoning, refer to [Martins 83].

and Sussman's work had two major influences in AI: (1) it opened a new perspective to the handling of alternatives (dependency-directed backtracking), and (2) it influenced the creation of systems that concentrate on how to handle contradictions (belief revision systems).

Building upon Stallman and Sussman's work, Doyle designed the *Truth-Maintenance System* (TMS) [Doyle 78, 79, 80], the first domain-independent belief revision system[3]. TMS maintains a knowledge base in which propositions are explicitly *marked* as believed or disbelieved. When a contradiction is found, TMS revises its beliefs so that no inconsistent propositions are believed.

TMS is based on the definition of two kinds of objects: *propositions* and *justifications*. Justifications represent the reasons why TMS believes or disbelieves a certain proposition. Attached to each proposition in the knowledge base, there are one or more justifications that support TMS's belief or disbelief in that proposition. Although Doyle points out the usefulness of four kinds of justifications [Doyle 79, pp.239-244], he mainly implemented one of them, the *SL-justifications*. This type of justification contains two lists of propositions, the *inlist* and the *outlist*. The proposition supported by an SL-justification is *believed* if and only if every proposition specified in the inlist is believed and every proposition specified in the outlist is disbelieved. Whenever a proposition is derived, it is justified by an SL-justification containing all the propositions *directly* used in its derivation and the rule of inference used to derive it.

In TMS, there are two distinguished types of propositions: (1) *Premises* are propositions whose SL-justification has empty inlist and empty outlist. A premise is always believed, (2) *Assumptions* are propositions whose SL-justification has empty inlist and non-empty outlist. Assumptions are propositions whose belief depends on the system's disbelief in other propositions (the propositions in its outlist).

---

[3] The field of belief revision in AI is usually recognized to have been initiated by the work of Doyle, although a system that performs belief revision (in robot planning) was developed simultaneously by Philip London [London 78].

TMS may be asked to add a new proposition to the knowledge base or to add or retract a justification for a proposition. In either case, TMS tries to find disbelieved propositions which will become believed by such addition or retraction and tries to find believed propositions that will become disbelieved by such addition or retraction.

In addition, TMS may be told that a proposition and its negation are both believed. In this case, the dependency-directed backtracking mechanism of Stallman and Sussman is invoked, which will search through the knowledge base, starting with the SL-justifications of the contradictory propositions, until it finds all the assumptions that are considered by the contradictory propositions. One of those assumptions is selected as the culprit for the contradiction and is *disbelieved*. To disbelieve this assumption, TMS believes in one of the propositions referenced in the outlist of the assumption and justifies this proposition with an SL-justification whose inlist contains the proposition representing the contradiction. After selecting the culprit for the contradiction, it is necessary to disbelieve all the propositions depending upon it. In TMS, this is done by following the chain of dependency records and disbelieving each proposition that has no SL-justification other than the one that includes the selected culprit in its inlist.

Doyle's research triggered the development of several belief revision systems [Goodwin 82, 84; McAllester 78, 80; McDermott 82, 83; Shrobe 79; Thompson 79]. These systems share two characteristics: (1) they are mainly concerned with implementation issues, paying no special attention to the logic underlying the system; (2) each proposition is justified with the propositions that directly originated it. The first aspect does not allow the formal study of the properties of the systems independently of their implementations: in those systems, it is very difficult to define and study the properties of the underlying logic except by repeatedly running the program[4]. The second aspect gives rise to systems that can only deal with one

[4]Although there are techniques to prove properties about programs, and thus one may be tempted to use them to prove properties about these programs, without the statement of the underlying logic one does not have a clear idea of what properties to prove.

situation at a time, are not able to perform inferences in a state where a contradiction was derived, and present a large computing overhead when switching between situations and in computing the culprit for a contradiction. A detailed description of these problems will be given in the next section.

As a reaction against the two problems mentioned, the early 80's saw the development of new research directions in belief revision systems, characterized by: (1) an explicit concern about the foundations of the systems, independently of their implementations [Doyle 82, 83; Martins 83; Martins and Shapiro 83] and (2) the use of a new type of justification [Martins 83; Martins and Shapiro 83; deKleer 84].

In this paper, we present a belief revision system based on a logic specifically conceived to support belief revision systems, discuss the properties of the system independently of its implementation (the abstract system), and present a particular implementation of the abstract model (SNeBR) using the SNePS Semantic Network Processing System [Shapiro 79a]. SNeBR is written in Franz Lisp and runs on VAX-11 systems at the Department of Computer Science, State University of New York at Buffalo and at the Instituto Superior Tecnico (School of Engineering of the Technical University of Lisbon, Portugal).

3. Assumption-Based vs Justification-Based Systems

A fundamental issue in belief revision systems is to be able to identify *every* proposition that may have contributed to a contradiction. This is important since, on the one hand, we don't want to blame some assumption irrelevant to the contradiction as the culprit, and, on the other hand, when looking for which assumption may be responsible for the contradiction we don't want to leave out any assumption possibly responsible for the contradiction. In order to do this, belief revision systems have to to keep a record of where each proposition in the knowledge base came from. These records are inspected while searching for the culprit of a contradiction. Thus, associated with every proposition in the knowledge base, there will be a set, called the *support* of the proposition, that tells where that proposition came from.

After selecting the culprit for a contradiction, the belief revision system typically wants to "change its beliefs", i.e., consider another set of the propositions in the knowledge base that does not contain the culprit for the contradiction nor any proposition derived from it. Furthermore, when considering a given set of propositions (beliefs), the belief revision system wants to ignore all the other propositions that may exist in the knowledge base.

In this section, we will discuss the alternative methods for recording the support of a proposition and evaluate each of them from the point of view of the actions that a belief revision system has to perform.

There are two different ways of recording the origin of propositions; corresponding to justification-based and assumption-based systems [deKleer 84]. In *justification-based* systems, the support of each proposition contains the propositions that *directly* originated it. This approach was taken by [Doyle 79; Goodwin 82, 84; McAllester 80; McDermott 82; Shrobe 79; Thompson 79]. In *assumption-based* systems, the support of each proposition contains the *hypotheses* (*non*-derived propositions) that originated it. This approach was taken by [Martins 83; deKleer 84].

In order to compare justification-based and assumption-based systems, let us consider the following example from [Charniak, Riesbeck and McDermott 80, p.197]: Suppose that the knowledge base contains the propositions: $(\forall x)[Man(x) \rightarrow Person(x)]$, $(\forall x)[Person(x) \rightarrow Human(x)]$, and $(\forall x)[Human(x) \rightarrow Person(x)]$. Adding Man(Fred) to the knowledge base will cause the derivation of Person(Fred) which, in turn, will cause the derivation of Human(Fred). Furthermore, the addition of Human(Fred) to the knowledge base causes Person(Fred) to be re-derived.

In a justification-based system, the support of each proposition contains the propositions that *directly* originated it. Under this approach, when Person(Fred) is derived from $\forall(x)[Man(x) \rightarrow Person(x)]$ and Man(Fred), its support will be {Man(Fred), $\forall(x)[Man(x) \rightarrow Person(x)]$}. Likewise, the support of Human(Fred) is {Person(Fred), $\forall(x)[Person(x) \rightarrow Human(x)]$}. Finally, when Person(Fred) is re-derived, its support will be

{Human(Fred), ∀(x)[Human(x)→Person(x)]}. In Figure 1, we represent the dependencies among the propositions in the knowledge base.

In this figure, two directed arcs (labeled **pr**, for premisses) pointing to a circle mean that the two propositions at the ends of the arcs were combined by some rule of inference, to produce the proposition that is pointed to by the arc (labeled **c**, for conclusion) leaving that circle. If there exists a path of arcs (alternatively labeled **pr** and **c**) from the proposition A to the proposition B then it means that the proposition B depends on proposition A. Notice that Human(Fred) depends on Person(Fred) which, in turn, depends on Human(Fred). This is called a *circular proof*.

In an assumption-based system, the support of each proposition contains the *hypotheses* (non-derived propositions) that originated it. Under this approach, when Person(Fred) is derived from ∀(x)[Man(x)→Person(x)] and Man(Fred), it is supported by these hypotheses,[5] i.e.,

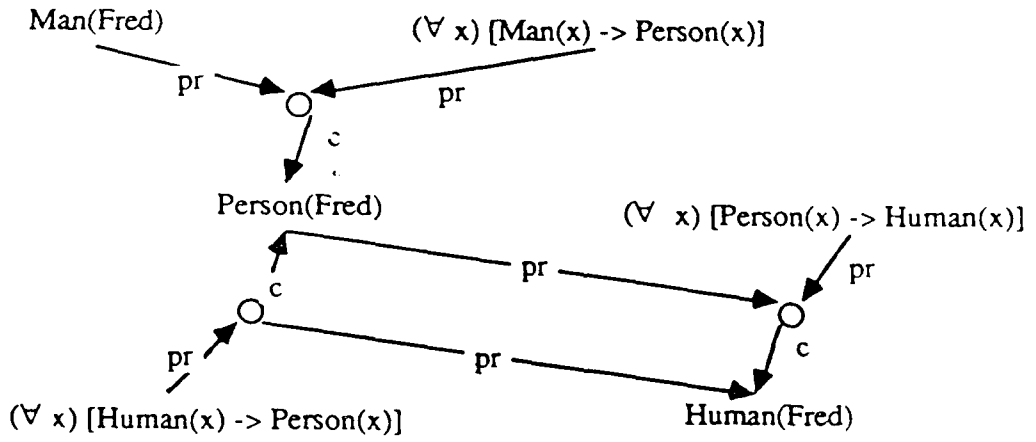


Figure 1
Knowledge base dependencies.
(justification-based systems)

its support is {Man(Fred), ∀(x)[Man(x)→Person(x)]}. When Human(Fred) is derived, it is supported by ∀(x)[Person(x)→Human(x)] and the hypotheses underlying Person(Fred); i.e., it is supported by {Man(Fred), ∀(x)[Man(x)→Person(x)], ∀(x)[Person(x)→Human(x)]}. Similarly, when Person(Fred) is re-derived it is supported with {Man(Fred), ∀(x)[Man(x)→Person(x)], ∀(x)[Person(x)→Human(x)], ∀(x)[Human(x)→Person(x)]}. Figure 2 shows the dependencies among the propositions in the knowledge base.

In this figure, a circle, called a *supporting node*, pointed to by an arc labeled **do** (derivation origin) represents the support for the proposition at the end of the arc. The arcs labeled **os** (origin set) leaving that circle point to the hypotheses from which the proposition was derived. Since each proposition is directly connected with the hypotheses that underlie it, there are no circular proofs.

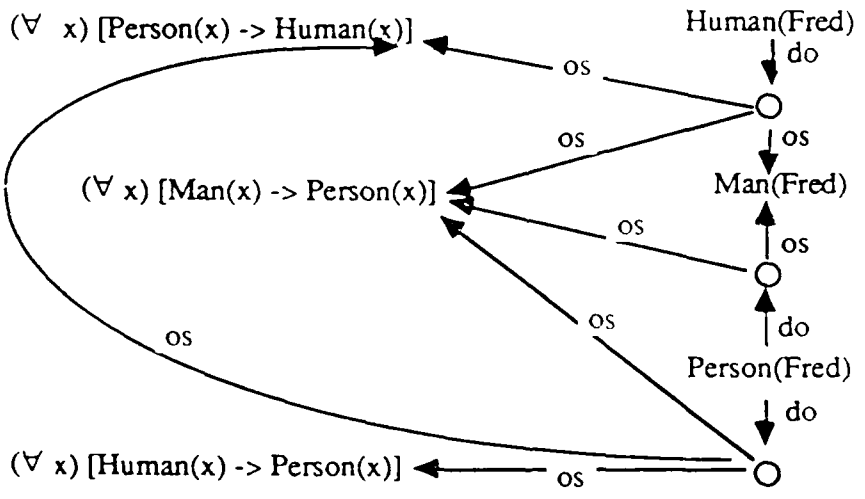


Figure 2
Knowledge base dependencies
(assumption-based systems)

[5]We are assuming that the propositions Man(Fred), ∀(x)[Man(x)→Person(x)], ∀(x)[Person(x)→Human(x)], and ∀(x)[Human(x)→Person(x)] are hypotheses, i.e., they were entered into the knowledge base rather than being derived.

Let us now evaluate both approaches with respect to the operations that are performed in a belief revision system. The first aspect that we will consider consists in *identifying the possible culprits for a contradiction*. We will assume that a contradiction is detected within the set of believed propositions in the knowledge base, say that both A and $\neg$A both belong to this set. To identify the possible culprits for the contradiction in justification-based systems, we will have to follow an arbitrarily long chain of arcs alternately labeled[6] c^c and pr^c until all the hypotheses (propositions with no incoming **c** arcs) underlying the contradictory propositions are identified. Furthermore, this arc-following procedure should be done with some care, due to the possible existence of circular proofs, or the program may be caught in an infinite loop. In assumption-based systems, the identification of the set of hypotheses underlying a contradiction is done in *exactly* two steps: traversal of the do^c leaving from each of the contradictory propositions and leading to the supporting nodes of the contradiction, and then the traversal of the os^c arc connecting the supporting node with the hypotheses underlying it. In other words, *in assumption-based systems it is easier to identify the possible culprits for a contradiction than in justification-based systems.*

The second aspect under which we will evaluate both approaches concerns the *computation of the support of each proposition* in the knowledge base. In justification-based systems, hypotheses have no support,[7] and thus when a new hypothesis is entered in the knowledge base all that has to be done is to add that hypothesis to the knowledge base. When a new proposition is derived, it is associated with the propositions that *immediately* originated it, and these propositions are known with no extra computation because they have just been used to derive the new proposition. Therefore, in justification-based systems, the creation of the support of a proposition is trivial. In assumption-based systems, hypotheses are associated with a support that points to the hypotheses themselves; When a proposition is derived, the belief

[6] If φ stands for the label of a directed arc, we will represent by φ^c the converse of φ.

[7] At least in the way we defined them. This may not be exactly true for existing systems (see for example [Doyle 79]).

revision system has to compute the support of the parent propositions, which, as we saw, is done by (1) following the do^c and os^c arcs, (2) from this set computing the support of the newly derived proposition, and (3) associating the computed support with the newly derived proposition. In conclusion, *in assumption-based systems, it is harder to compute supports than in justification-based systems.*

Let us now evaluate each of these approaches from the standpoint of the *computation of the set of believed propositions* and *deciding which propositions should be considered for inference*. We said that one of the roles of a belief revision system consists of hiding from the inference system some of the propositions in the knowledge base, namely all those propositions that one does not want to consider any longer. Let us assume for the sake of argument that we decided that the proposition $\forall(x)[Person(x) \rightarrow Human(x)]$ be removed from the knowledge base. Clearly this entails that Human(Fred), which was derived from it, should be removed from the knowledge base as well. This removal originates a new knowledge base in which only the propositions Man(Fred), $\forall(x)[Man(x) \rightarrow Person(x)]$, $\forall(x)[Human(x) \rightarrow Person(x)]$ and Person(Fred) should be considered.

There are two ways to obtain this new knowledge base in a justification-based system; (1) We may physically erase from the knowledge base all the propositions no longer wanted. This approach is not the most appropriate, since if later on we re-consider an erased proposition we also want to add to the knowledge base all the propositions that were derived from it, and the only way of doing this is by actually re-deriving them. (2) The other alternative is to tell the knowledge-base retrieval function which propositions should and should not be considered at any moment. This permits some savings when a proposition once believed but later disbelieved is believed once more. In justification-based systems, this second approach can be obtained by actually *marking* the propositions that should not be considered by the knowledge-base retrieval function. In Figure 3, we show the knowledge base, marking with a "*" every proposition that should not be considered by the knowledge-base retrieval function.

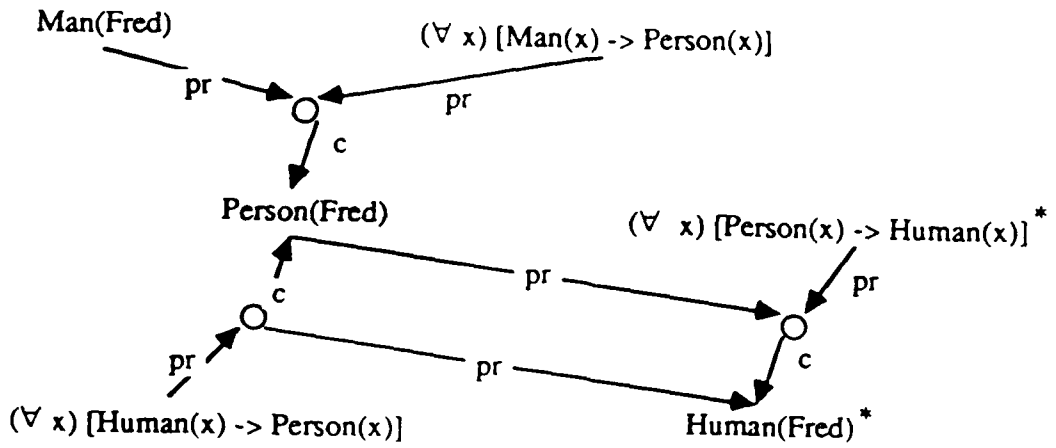


Figure 3
Knowledge base after removal of ∀(x)[Person(x)→Human(x)]
(justification-based systems)

When the retrieval function considers the propositions in the knowledge base, it should check whether or not there is a * attached to propositions before retrieving them. Under this approach, when a proposition is removed from the knowledge base, the belief revision system has to go through the knowledge base[8] deciding what the consequences of the removal are and "marking" propositions (attaching a * to them). A similar procedure has to take place if we decide to re-consider some proposition that is marked with a *: the knowledge base has to be searched to decide which *-ed propositions should be "unmarked". This is the approach taken in the existing justification-based systems, in which *changing beliefs is difficult*.[9]

Let us consider how this can be done in an assumption-based system. In these systems, every proposition is supported by hypotheses. Assuming that the knowledge base retrieval function knows which hypotheses are under consideration, whenever it "looks" into the knowledge base it can find out whether or not some proposition should be considered just by looking at the hypotheses in its support. Notice that these hypotheses are *directly linked* to the

[8]Following the c^c/pr^c path.

[9]In the existing implementations of justification-based systems, for example TMS, the only way to switch between contexts is by the introduction of a contradiction. This problem, however, is not inherent to justification-based systems.

proposition, and so this operation is not very difficult or expensive to perform. In other words, in assumption-based systems there is no marking of the propositions in the knowledge base; it is the knowledge-base retrieval function that decides dynamically[10] which propositions should be considered. Thus, in assumption-based systems, *changing beliefs is easy* — it corresponds to giving a different argument to the knowledge-base retrieval function.

Thus, in justification-based systems there exists one global set of beliefs at any moment (the set of beliefs is characterized by containing every proposition in the knowledge base that does not have a *. Changing sets of beliefs entails an inspection of the entire knowledge base. In assumption-based systems, there is no global set of beliefs. The number of sets of beliefs in the knowledge base is the power set of the hypotheses in the knowledge base, and the propositions to be considered at any given moment are dynamically selected by the knowledge-base retrieval function.

In justification-based systems, there is another problem that arises during the process of changing sets of beliefs, which is related to the existence of circular proofs. Suppose that, given the knowledge base represented in Figure 1, we decide that Man(Fred) and all the propositions depending upon it should be removed from the knowledge base. The dependency arc leaving Man(Fred) leads to Person(Fred). However, Person(Fred) has another support, and one is faced with the problem of whether or not to remove Person(Fred) from the knowledge base, since although one of its supports is no longer valid, Person(Fred) may still be in the knowledge base due to the other support. This problem forces the designers of justification-based systems to decide whether or not circular proofs should be recorded in the knowledge base. If they are, the process of changing beliefs may become very complex, possibly requiring several passes through the knowledge base until all circularities are resolved; if they are not recorded, then whenever some proposition is about to be removed from the knowledge base, the

[10] every time it performs a knowledge base retrieval.

belief revision system should check whether or not there is an alternative derivation for that proposition by actually trying to derive it again in the current knowledge base. A discussion of the possible solutions to these problems can be found in [Doyle 79] and [Charniak, Riesbeck and McDermott 80]. This problem does not arise in assumption-based systems, in which circular proofs do not exist.

Another important issue in belief revision systems concerns how to compare results obtained from different sets of beliefs, i.e., how to compare different alternatives. In justification-based systems, this represents a difficult problem due to their notion of a single set of beliefs: at any given moment there is a set of propositions that is being considered, and if one wants to consider a different set of propositions, then the knowledge base *has to be changed* through the marking/unmarking process. In assumption-based systems, as we saw, there are multiple sets of beliefs simultaneously represented, and thus it is easy to compare results obtained in different sets of beliefs.

In conclusion, assumptions-based systems present several advantages over justification-based systems, with respect to: (1) Identifying the possible culprits for a contradiction, (2) changing sets of beliefs, and (3) comparing sets of beliefs. The main advantage that justification-based systems present over assumption-based ones concerns the explanation of their reasoning. In fact, since these systems maintain a record of the history of the derivation of each proposition in the knowledge base they can explain how a given proposition was obtained. DeKleer [deKleer 84] proposes a system which records both assumptions and justifications.

However, there is a hidden assumption behind our discussion, which is that we are able to compute *exactly* which hypotheses underlie a given proposition, *nor more nor less*. The obvious solution of unioning the hypotheses underlying each of the parent propositions to com-

pute the hypotheses underlying a derived proposition won't do.[11] In some cases, we have to perform a set difference rather than a union;[12] in some other cases, we may even forbid the combination of two propositions if that combination can lead us to trouble.[13] Another issue that remains unexplored is how to "remember" the contradictions that were derived and thus avoid getting into the same contradiction twice.

In the next section we present a logic, the SWM system, that addresses these two problems. Each proposition in SWM is associated with a set (the origin set) that contains those, *and only those*, hypotheses used in its derivation. Each proposition in SWM is also associated with a set (the restriction set) containing the sets of hypotheses which are incompatible (produce inconsistencies) with the proposition's origin set. The SWM system defines how these sets are formed and propagate through the application of the rules of inference. Based on SWM, we define an abstract model for an assumption-based belief revision system and describe a particular implementation of this abstract model.

4. Theoretical Foundations - The SWM System

In this section we introduce a logic, the SWM[14] system, that was developed to support belief revision systems. When discussing a logic, there are two aspects to consider, its syntax and its semantics.

The *syntax* of a logic includes a set of formation rules and a set of rules of inference. The set of *formation rules* determines which formulas are legal in the logic. These formulas

---

[11]This is implicitly done in some justification-based systems, e.g., with the SL-justifications of TMS [Doyle 79].

[12]Assume that B is a hypothesis and that A has support $\{B\} \cup \alpha$. It should be clear that the proposition $B \rightarrow A$ should be derivable; furthermore, $B \rightarrow A$ should not depend on B. As we will see in the next section, its support will be α.

[13]Suppose that A has support α and that B has support β. The question that is raised is what should the support of $A \wedge B$ be? $\alpha \cup \beta$? In this case, from $A \wedge B$ we certainly can get either A or B, whose support then would be $\alpha \cup \beta$, i.e., it will contain hypotheses not *used* in the derivation of the wff. The right decision is not to allow the derivation of $A \wedge B$ if A and B have different supports. A complete discussion on this issue can be found in [Martins and Shapiro forthcoming].

[14]After Shapiro, Wand and Martins. The SWM system is a successor of the system of [Shapiro and Wand 76], which was modified to cope with contradictions.

are called well-formed formulas, *wffs* for short. We will assume standard formation rules for wffs with $\neg$, $\vee$, $\wedge$, $\rightarrow$ as connectives and $\forall$, $\exists$ as quantifiers. See, for example, [Lemmon 78, pp.44 and 104]. The set of *rules of inference* (the deductive system) specifies which conclusions may be inferred from which premises. Given an argument (P,c),[15] we say that c is *deducible* from P, written $P \vdash c$, if there is a sequence of rules of inference which when applied to P produces c.

The *semantics* of a logic concerns the study of the conditions under which sentences are true or false. The semantics are completely determined by the specification of two things, the *interpretations* of the language (every possible assignment of a particular object to each particular member of the language) and the *truth conditions* for it (what it means for a given sentence to have a given truth value in a given interpretation). We say that the argument (P,c) is *valid* if there is no interpretation in which each sentence in P is true and in which c is false. If (P,c) is valid, we write $P \models c$.

There is nothing about validity in the deductive system, and there is nothing about deducibility in the semantics. Although syntax and semantics are separate parts of a logical system, and thus deducibility and validity are intensionally distinct, they must fit together properly in order for the system to make any sense. A logic is said to be *sound* if and only if every argument deducible in its deductive system is valid according to its semantics. A logic is said to be *complete* if and only if every argument valid according to its semantics is deducible in its deductive system. Given a "reasonable" semantics, a logic can be unsound due to "wrong" rules of inference; and a logic can be incomplete due to the lack of necessary rules of inference or due to rules of inference that are too constraining. The SWM system is an incomplete logic, since several arguments valid according to its semantics are not deducible in its deduction system. This fact should not be regarded as a drawback of the logic but rather as a

[15] A premisse-conclusion argument is an ordered pair (P,c) in which P is a set of propositions, called *premisses* and c is a single proposition, called *conclusion*.

feature that makes it attractive for its intended applications.

The first step towards formally analyzing arguments consists of providing precise meaning for everyday terms like "and", "or", "if", "if...then...", "every", "some", etc. In the process of translating an informal argument into a formal one, some of the features of the informal argument are lost. As Haack says:

> Some informal arguments are intuitively judged to be valid, others invalid. One then constructs a formal language in which the relevant structural features of those arguments can be schematically represented, and axioms/rules which allow the intuitively approved, and disallow the intuitively disapproved arguments. . . . However, if formal logic faithfully followed informal arguments in all their complexity and vagueness there would be little point in formalisation . . . but considerations of simplicity, precision and rigour may be expected to lead to discrepancies between informal arguments and their formal representations . . . One should recognise, then, that a failure on the part of a formal system to represent all the knobs and bumps of the informal arguments it systematises is not necessarily objectionable. On the other hand, one must be wary of assuming that all adjustments are acceptable; one needs to ask whether the gains in simplicity and generality compensate for the discrepancy. [Haack 78, pp.32-34][17]

The important point is to keep in the model those features that are of interest to the modeler. Therefore, when assigning meaning to the logical terms, one should bear in mind which features of the informal arguments one wants to preserve in their formal counterparts. In our case, our main goal is to keep a record of propositional dependencies, and our approach adopts the meaning of the logical connectives used in classical logic and builds a deductive system that blocks some unwanted deductions (resulting in an incomplete system). Most of the blocked deductions involve the introduction of irrelevancies.

Before presenting the SWM system, let us informally discuss what types of information we need in our logic.[18] One of the fundamental problems that any logic underlying a belief revision system has to address is how to keep track of and propagate propositional dependencies. This is important, because, in the event of detection of a contradiction, one should be able

---

[17] The above quote may be taken as an argument against formal systems/logic in AI. However, we should bear in mind that models are always simplifications of the phenomenon being modeled and therefore that some features are always lost during the modeling process. Systems that are programmed model, in a formal and thus subject to these considerations.

[18] A detailed discussion of these issues can be found in [Martins 83] and [Martins and Shapiro, forthcoming].

to identify *exactly which* assumptions were used in the derivation of the contradictory propositions. We don't want to blame some assumption irrelevant to the occurrence of the contradiction as the culprit for the contradiction, and, when looking for the possible culprits for a contradiction, we don't want to leave out any assumption possibly responsible for the contradiction. In the field of logic, the relevance logicians[18] also want to keep track of what propositions were used to derive any given proposition. Relevance logicians have developed mechanisms to keep track of what assumptions were used in the derivation of a given proposition and to prevent the introduction of irrelevancies. One way of doing this (used in the FR system of [Anderson and Belnap 75, pp.346-348] and in the system of [Shapiro and Wand 76]) consists of associating each wff with a set, called the *origin set*, which references every hypothesis used its derivation. The rules of inference are stated so that all the wffs derived using a particular hypothesis will reference this hypothesis in their origin sets. Whenever a rule of inference is applied, the origin set of the resulting wff is computed from the origin sets of the parent wffs.[19] In order to guarantee that the origin set only contains the hypotheses actually used in the derivation of the wff, and no more hypotheses, some of the applications of the rules of inference allowed in classical logic are blocked. Most of this mechanism was adopted in the SWM system.

Besides the dependency-propagation mechanism, there is another advantage in using relevance logic, to support belief revision systems. In classical logic a contradiction implies anything; thus, in a belief revision system based on classical logic, whenever a contradiction is derived it should be discarded immediately. In a relevance-logic-based belief revision system, we may allow the existence of a contradiction in the knowledge base without the danger of filling the knowledge base with unwanted deductions. In a relevance logic-based belief revision system all a contradiction indicates is that any inference depending on *every* hypothesis

[18] Appendix 2 presents an introduction to relevance logic.

[19] The resulting origin set can either be the union of the origin sets of the parent wffs or the set difference of the origin sets of the parent wffs.

underlying the contradiction is of no value. In this type of systems we can even perform reasoning in a knowledge base known to be inconsistent.[20]

Another important issue in belief revision systems which will be reflected in our logic consists in the recording of the conditions under which contradictions may occur. This is important because once we discover that a given set is inconsistent,[21] we may not want to consider it again, and even if we do want to consider it, we want to keep in mind that we are dealing with an inconsistent set. In the SWM system, contradictions are recorded by associating each wff with a set, called the *restriction set*, that contains information about which sets unioned with the wff's origin set produce an inconsistent set. When new wffs are derived, their restriction sets are computed directly from the restriction sets of the parent wffs, and when contradictions are detected all the wffs whose origin set references any of the contradictory hypotheses has its restriction set updated in order to record the newly discovered contradictory set. Similarly to what happens with origin sets, we will make sure that restriction sets don't have any more information than they should.

In addition, for the proper application of some rules of inference, it is important to know whether a given wff was introduced as a hypothesis or was derived from other wffs. In order to do this, we associate each wff with an identifier, called the *origin tag* that tells whether the wff is a hypothesis, a normally derived proposition or a special proposition, that if treated regularly, would introduce irrelevancies into the knowledge base.[22]

Formally, the SWM system deals with objects called *supported wffs*. A supported wff consists of a wff and an associated triple containing an *origin tag* (OT), an *origin set* (OS), and a *restriction set* (RS). The set of all supported wffs is called the *knowledge base*. We write

---

[20] See, for example, [Martins 83]

[21] A set is *inconsistent* if a contradiction may be derived from it. A set is *consistent* just in case it is not inconsistent. We represent a contradiction by $\rightarrow\leftarrow$, thus A is inconsistent if $A \vdash \rightarrow\leftarrow$

[22] For a discussion of this latter case and the reasons that lead us to introduce this additional value for origin tags, refer to [Martins 83] or [Martins and Shapiro forthcoming]

$A \mid \tau,\alpha,\rho$ to denote that A is a wff with OT τ, OS α, and RS ρ, and we define the functions **ot**(A)=τ, **os**(A)=α and **rs**(A)=ρ.[23] The OS is a set of hypotheses. The OS of a supported wff contains those (and only those) hypotheses that were *actually used* in the derivation of that wff. The OTs range over the set {hyp, der, ext}: *hyp* identifies hypotheses, *der* identifies normally derived wffs within SWM, and ***ext*** identifies special wffs whose OS was extended. An RS is a set of sets of wffs. A wff, say A, whose RS is $\{R_1, \ldots, R_n\}$ means that the hypotheses in **os**(A) added to any of the sets $R_1, \ldots, R_n$ produces an *inconsistent set*. The RS of an extended wff will contain *every* set which unioned with the wff's OS will produce a set that is known to be inconsistent. Our rules of inference guarantees that the information contained in the RS is carried over to the new wffs whenever a new proposition is derived. Furthermore, the rule of inference guarantee (see the theorems stated in Appendix 1) that RSs do not contain any redundant information; i.e., given $A \mid \tau,\alpha,\{R_1, \ldots, R_n\}$, the following types of redundancy do not arise:

1. There is no $\rho \in \{R_1, \ldots, R_n\}$ such that $\rho \cap \alpha = \emptyset$.[24]

2. There are no $\mu \in \{R_1, \ldots, R_n\}$ and $\nu \in \{R_1, \ldots, R_n\}$, such that $\mu \subset \nu$.[25]

We say that the supported wff $A \mid \tau,\alpha,\{R_1, \ldots, R_n\}$ has a *minimal* RS if the following two conditions are met:

1. $\forall r \in \{R_1, \ldots, R_n\}\ (r \cap \alpha = \emptyset)$;

2. $\forall r,s \in \{R_1, \ldots, R_n\}\ r \not\subset s$.

In Appendix 1, we prove that all the supported wffs in the knowledge base resulting from the application of the rules of inference of the SWM system have minimal RS.

---

[23] We will discuss the problem of multiple derivations of the same wff later in this paper (Section 7).

[24] Otherwise, the set would contain extra information, namely, all the wffs in $\rho - \alpha$.

[25] Otherwise, the set ν could be discarded from the restriction set without any loss of information. Since μ belongs to the RS, we know that that $O_{\alpha\cup\mu} \vdash \rightarrow\!\!\!

To compute the RS of a wff resulting from the application of the rules of inference, we define the functions μ and $\int$.[26] The function μ is used whenever a rule of inference which generates a supported wff whose OS is the union of the OSs of the parent wffs is applied. It generates the RS of the resulting wff by unioning the RSs of the parent wffs and removing from the resulting set some sets which would be redundant, namely that would violate one of the two conditions listed above. The function $\int$ is used by the rules of inference which generate a supported wff with a smaller OS than the parent wffs. It takes the RS of the several hypotheses in the resulting OS and computes a minimal RS from those RSs. The functions μ and $\int$ are defined as follows:

$$\mu(\{r_1,\ldots,r_m\},\{o_1,\ldots,o_n\}) = \sigma(\Psi(r_1\cup\ldots\cup r_m, o_1\cup\ldots\cup o_n)),$$

where

$$\Psi(R,O) = \{\alpha : (\alpha\in R \wedge \alpha\cap O=\emptyset) \vee (\exists\beta\in R)[\beta\cap O\neq\emptyset \wedge \alpha=\beta-O]\}$$

and

$$\sigma(R) = \{\alpha\in R \quad \neg(\exists\beta)[\beta\neq\alpha \wedge \beta\in R \wedge \beta\subset\alpha]\}$$

and

$$\int(O) = \mu(\{r:\exists H\in O: r=\mathbf{rs}(H)\},\ \{o:\exists H\in O: o=\mathbf{os}(H)\})$$

To compute the OT of a wff resulting from the application of the rules of inference, we define the function Λ as follows:

$$\Lambda(\alpha,\beta) = \begin{cases} ext & \text{if } \alpha=ext \text{ or } \beta=ext \\ der & \text{otherwise} \end{cases}$$

$$\Lambda(\alpha,\beta,\ldots,\gamma) = \Lambda(\alpha,\Lambda(\beta,\ldots,\gamma))$$

Two supported wffs are said to be *combinable* by some rule of inference if the supported wff resulting from the application of the rule of inference has an OS that is not known to be inconsistent. We define the predicate *Combine*, which decides the combinability of the sup-

---

[26] The "integration sign" is used because this function "integrates" the information contained in the RSs of a set of hypotheses.

ported wffs A and B:

$$\text{Combine}(A,B) = \begin{cases} \textit{false} & \text{if } \exists r \in \mathbf{rs}(A) : r \subset \mathbf{os}(B) \\ \textit{false} & \text{if } \exists r \in \mathbf{rs}(B) : r \subset \mathbf{os}(A) \\ \textit{true} & \text{otherwise} \end{cases}$$

The rules of inference of the **SWM system**, guarantee that:

1. The OS of a supported wff contains *every* hypothesis that was used in its derivation.
2. The OS of a supported wff *only* contains the hypotheses that were used in its derivation.
3. The RS of a supported wff records *every* set known to be inconsistent with the wff's OS.
4. The application of rules of inference is blocked if the resulting wff would have an OS known to be inconsistent.

It is important to distinguish between a set *being* inconsistent and a set *being known to be* inconsistent. An inconsistent set is one from which a contradiction *can be* derived; a set known to be inconsistent is an inconsistent set from which a contradiction *has been* derived. The goal of adding RSs is to avoid re-considering *known* inconsistent sets of hypotheses.

The OT and OS of a proposition reflect the way the proposition was derived: the OS contains the hypotheses underlying that proposition, and the OT represents the *relation* between the proposition and its OS. The RS of a proposition reflects our current knowledge about how the hypotheses underlying that proposition relate to the other hypotheses in the knowledge base. Once a proposition is derived, its OT and OS remain constant; however, its RS changes as the knowledge about all the propositions in the knowledge base does.

The following are the rules of inference of the SWM system.[27]

[27] There is an extra connective in the SWM system, the **truth-functional or**, which will not be discussed in this paper. For a detailed description of this connective, refer to [Martins 83], [Martins and Shapiro 84] or [Martins and Shapiro forthcoming].

Hypothesis (Hyp): For any wff A and sets of wffs $R_1 \ldots R_n$ ($n \geq 0$), such that $\forall r \in \{R_1, \ldots, R_n\}$: $r \cap \{A\} = \emptyset$ and $\forall r,s \in \{R_1, \ldots, R_n\}$: $r \subset s$, we may add the supported wff $A \mid hyp,\{A\},\{R_1, \ldots, R_n\}$ to the knowledge base, provided that A has not already been introduced as a hypothesis.

Implication Introduction ($\rightarrow$I): From $B \mid der,o,r$ and any hypothesis $H \in o$, infer $H \rightarrow B \mid der,o-\{H\},(o-\{H\})$.

Modus Ponens — Implication Elimination, Part 1 (MP): From $A \mid t_1,o_1,r_1$, $A \rightarrow B \mid t_2,o_2,r_2$, and Combine(A,A$\rightarrow$B), infer $B \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

Modus Tollens — Implication Elimination, Part 2 (MT): From $A \rightarrow B \mid t_1,o_1,r_1$, $\neg B \mid t_2,o_2,r_2$, and Combine($A \rightarrow B$,$\neg B$), infer $\neg A \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

Negation Introduction ($\neg$I): From $A \mid t_1,o,r$, $\neg A \mid t_2,o,r$, and any set $\{H_1, \ldots, H_n\} \subset o$, infer $\neg(H_1 \wedge \ldots \wedge H_n) \mid \Lambda(t_1,t_2),o-\{H_1, \ldots, H_n\},(o-\{H_1, \ldots, H_n\})$.

From $A \mid t_1,o_1,r_1$, $\neg A \mid t_2,o_2,r_2$, $o_1 \neq o_2$, Combine(A,$\neg$A), and any set $\{H_1, \ldots, H_n\} \subset (o_1 \cup o_2)$, infer $\neg(H_1 \wedge \ldots \wedge H_n) \mid ext, (o_1 \cup o_2)-\{H_1, \ldots, H_n\}, ((o_1 \cup o_2)-\{H_1, \ldots, H_n\})$.

Negation Elimination ($\neg$E): From $\neg\neg A \mid t,o,r$, infer $A \mid \Lambda(t,t),o,r$.

Updating of Restriction Sets (URS): From $A \mid t_1,o_1,r_1$, and $\neg A \mid t_2,o_2,r_2$, we **must** replace each hypothesis $H \mid hyp,\{H\},R$ such that $H \in (o_1 \cup o_2)$ by $H \mid hyp,\{H\},\sigma(R \cup \{(o_1 \cup o_2)-\{H\}\})$. Furthermore, we **must** also replace every supported wff $F \mid t,o,r$ (t=der or t=ext) such that $o \cap (o_1 \cup o_2) \neq \emptyset$ by $F \mid t,o,\sigma(r \cup \{(o_1 \cup o_2)-o\})$.

And Introduction ($\wedge$I): From $A \mid t_1,o,r$ and $B \mid t_2,o,r$, infer $A \wedge B \mid \Lambda(t_1,t_2),o,r$.

From $A \mid t_1,o_1,r_1$, $B \mid t_2,o_2,r_2$, $o_1 \neq o_2$, and Combine(A,B), infer $A \wedge B \mid ext,o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

And Elimination ($\wedge$E): From $A \wedge B \mid t,o,r$, and $t \neq ext$, infer either $A \mid der,o,r$ or $B \mid der,o,r$ or both.

Or Introduction ($\vee$I): From $\neg A \rightarrow B \mid t_1,o,r$ and $\neg B \rightarrow A \mid t_2,o,r$, infer $A \vee B \mid \Lambda(t_1,t_2),o,r$.

Or Elimination ($\vee$E):

From $A \vee B \mid t_1,o_1,r_1$, $\neg A \mid t_2,o_2,r_2$, and Combine($A \vee B, \neg A$), infer $B \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

From $A \vee B \mid t_1,o_1,r_1$, $\neg B \mid t_2,o_2,r_2$, and Combine($A \vee B, \neg B$), infer $A \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

From $A \vee B \mid t_1,o_1,r_1$, $A \rightarrow C \mid t_2,o_2,r_2$, $B \rightarrow C \mid t_3,o_2,r_2$, and Combine($A \vee B, A \rightarrow C$), infer $C \mid \Lambda(t_1,t_2,t_3),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$.

$\forall$ **introduction** ($\forall$I): From $B(t) \mid der,o \cup \{A(t)\},r$, in which $A(t)$ is a hypothesis which uses a term (t) never used in the system prior to A's introduction, infer $\forall(x)[A(x) \rightarrow B(x)] \mid der,o,(o)$.[28]

$\forall$ **elimination - Universal Instantiation** ($\forall$E): From the supported wffs $\forall(x)[A(x) \rightarrow B(x)] \mid t_1,o_1,r_1$, $A(c) \mid t_2,o_2,r_2$ and Combine($\forall(x)[A(x) \rightarrow B(x)], A(c)$), in which c is any individual symbol, infer $A(c) \rightarrow B(c) \mid \Lambda(t_1,t_2),o_1 \cup o_2,\mu(\{r_1,r_2\},\{o_1,o_2\})$;

$\exists$ **introduction** ($\exists$I): From $A(c) \mid t,o,r$ in which c is an individual constant, infer $\exists(x)[A(x)] \mid \Lambda(t,t),o,r$;

$\exists$ **elimination** ($\exists$E): From $\exists(x)[A(x)] \mid t,o,r$ and any individual constant c which was never used before, infer $A(c) \mid \Lambda(t,t),o,r$.

---

[28] According to this rule of inference, the universal quantifier can only be introduced in the context of an implication. This is not a drawback, as may seem at first, since the role of the antecedent of the implication ($A(x)$) is to define the type of object that are being quantified. This is sometimes called relativized quantification.

Among others, the following theorems hold for SWM (their proof can be found in Appendix 1 — the numbers associated with the theorems relate to the numbers in Appendix 1):

Theorem 4: All the supported wffs in the knowledge base resulting from the application of the rules of inference of SWM have minimal RS.

Theorem 5: In the knowledge base resulting from the application of the rules of inference of SWM, if two supported wffs have the same OS, then they have the same RS as well.

Corollary 5.1: Every OS has recorded with it *every* known inconsistent set.

5. Non-Standard Connectives - The SNePS Connectives

In Section 4, we were concerned with formally defining a logic that would not allow the introduction of irrelevancies and would be able to deal with contradictions. Here we are concerned with using such a logic in practical applications. To do so, we extend SWM by adding non-standard connectives (${}_n\mathbb{X}_i^j$, ${}_n\Theta_i$, $\vee\!\!\rightarrow$, $\wedge\!\!\rightarrow$). These non-standard connectives [Shapiro 79a, 79b] were originally motivated by the issues in knowledge representation, and an interest in carrying out deductions within the representation formalism, rather than about it. They were originally implemented in SNePS using a standard logic [Shapiro 79a]; we shall refer to them as the *SNePS connectives*.[26]

This discussion does not mean that we chose a "wrong" set of connectives when defining SWM. The connectives that were defined in Section 4 have simple semantics and make the task of talking about the properties of the logic easier. However, they lack some expressive power, and this is the reason for the introduction of the SNePS connectives. A second point in

[26] In this section, we expand our set of formation rules with the addition of the following rules for creating wffs: If $A_1, \ldots, A_n, C_1, \ldots, C_m$ are wffs, then $(A_1, \ldots, A_n) \wedge\!\!\rightarrow (C_1, \ldots, C_m)$ and $(A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m)$ are wffs; If $A_1, \ldots, A_n$ are wffs, $i \geq 0$, $i \leq j$ and $j \leq n$ then ${}_n\mathbb{X}_i^j(A_1, \ldots, A_n)$ is a wff; If $A_1, \ldots, A_n$ are wffs and $0 \leq i \leq n$ then ${}_n\Theta_i(A_1, \ldots, A_n)$ is a wff.

favor of having defined standard connectives in SWM is that the SNePS connectives are defined in this section in terms of the standard ones, and therefore a formula using the SNePS connectives should be considered as a *syntactic abbreviation* of a complex formula using the standard connectives.

We defined the term "rule of inference" to denote those statements in the meta-language that are used to add new lines to a proof; i.e., rules of inference are tools for lengthening deductions. In this section, we introduce the concept of a deduction rule. A *deduction rule* is defined to be any proposition that has either a connective or a quantifier (or both). A deduction rule is a statement in the object language, and can be considered a recipe, plan, or heuristic for deriving new information from old information. As an example, let us consider the wff A→B, as a deduction rule. This wff may be used to derive B, if A is known. From an applications point of view, we can look at this wff as stating something like "if you want to show B, then try to show A first"; i.e., we can reduce the problem of showing B to the problem of showing A.[30] From this deduction rule and the knowledge of A, the rule of inference MP enables the derivation of B. Rules of inference are entities that are part of a logic; deduction rules are wffs that are created while using the logic that contain some characteristics pertaining to the specific domain of application of the logic.

To make the rules of inference for the SNePS connectives easier to state, we introduce the following notation:

1. Let $C_i^n(a_1, \ldots, a_n)$ represent an ordered set (using some ordering criteria which we don't care about) of all the i-combinations of the elements of the set $\{a_1, \ldots, a_n\}$. We represent the k-th element of $C_i^n(a_1, \ldots, a_n)$ by ${}_kc_i^n(a_1, \ldots, a_n)$, and the set $\{a_1, \ldots, a_n\} - {}_kc_i^n(a_1, \ldots, a_n)$ by ${}_k\bar{c}_i^n(a_1, \ldots, a_n)$. For notational convenience, when the arguments of ${}_kc_i^n$ and ${}_k\bar{c}_i^n$ are left unspecified they will default to $P_1, \ldots, P_n$. For exam-

[30] Clearly, from the way → was defined, this rule can also be used to prove ¬A.

ple, $C_2^3(A,B,C) = \{\{A,B\}, \{A,C\}, \{B,C\}\}$, ${}_1c_2^3(A,B,C)=\{A,B\}$, ${}_2c_2^3(A,B,C)=\{A,C\}$, and ${}_2\bar{c}_2^3(A,B,C)=\{B\}$. Also, $C_2^3=\{\{P_1,P_2\},\{P_1,P_3\},\{P_2,P_3\}\}$, and ${}_2\bar{c}_2^3=\{P_2\}$.

2. We write $\varphi\{P_1, \ldots, P_n\}$ to denote the function application $\varphi(P_1, \ldots, P_n)$, and we write $\varphi<\{P_1, \ldots, P_n\}>$ to denote the set $\{\varphi(P_1), \ldots, \varphi(P_n)\}$. For example, $\wedge\{A_1, \ldots, A_n\} = A_1 \wedge \ldots \wedge A_n$; and $\neg<\{A_1, \ldots, A_n\}> = \{\neg A_1, \ldots, \neg A_n\}$.

The function Combine is extended in the following way to decide the combinability of more than two wffs:

$$\text{Combine}(A_1, \ldots, A_n) = \begin{cases} \textit{false} & \text{if } \exists r \in \mathcal{U}[\mathbf{rs}<\{A_1 \cdots A_n\}>, \mathbf{os}<\{A_1 \cdots A_n\}>]: \\ & \quad r \subset \mathcal{U}[\mathbf{os}<\{A_1 \cdots A_n\}>] \\ \textit{true} & \text{otherwise} \end{cases}$$

5.1. And-Entailment

And-entailment is a generalization of SWM's entailment to take two sets of arguments, a set of antecedents and a set of consequents. And-entailment, written $\wedge\rightarrow$, takes as arguments two sets of propositions. The proposition represented by the wff $(A_1, \ldots, A_n) \wedge\rightarrow (C_1, \ldots, C_m)$ asserts that the conjunction of the antecedents $(A_1, \ldots, A_n)$ relevantly[31] implies the conjunction of the consequents $(C_1, \ldots, C_m)$.

Before presenting the introduction and elimination rules for and-entailment, we should say that, as opposed to SWM's entailment, there is only one rule for eliminating $\wedge\rightarrow$, corresponding to a generalization of MP. The reason for this is related to the distinction between a deduction rule and a rule of inference. Recall that a deduction rule is taken to be a recipe to obtain new information from old information. From this point of view, the wff $(A_1, \ldots, A_n) \wedge\rightarrow (C_1, \ldots, C_m)$ is the formalization of the following heuristic: "*if you want to obtain any one of* $C_1, \ldots, C_m$ *then try to obtain all of* $A_1, \ldots, A_n$". In other words, the consequents of an and-entailment are propositions that, from the nature of the problem domain

[31]The condition that there is a *relevant connection between antecedents and consequents* represents a departure from the way the SNePS connectives were defined in [Shapiro 79a, 79b].

(whatever it is), typically are derived propositions as opposed to statements entered in the system as data. From this point of view, applying MT to this deduction rule doesn't make much sense, since it corresponds to trying to obtain information about the A's by looking for information about the C's, but, from the nature of the problem the C's are known only if the A's are.

The introduction and elimination rules for $\wedge\rightarrow$ are formalized as follows:

And-Entailment Introduction ($\wedge\rightarrow$I): from $(C_1\wedge\ldots\wedge C_m)$ | der,o$\cup\{A_1,\ldots,A_n\}$,r infer $(A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ | der,o,r-(o).

And-Entailment Elimination ($\wedge\rightarrow$E): From $(A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ | t,o,r, A_1 | t_1,o_1,r_1, ..., A_n | t_n,o_n,r_n, and Combine$((A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m),A_1,\ldots,A_n)$ infer $C_1\wedge\ldots\wedge C_m$ | $\Lambda(t,t_1,\ldots,t_n)$,o$\cup\{O\}$,$\mu(R,O)$, where O and R are the sets $O=\{o,o_1,\ldots,o_n\}$ and $R=\{r,r_1,\ldots,r_n\}$.

The rule of $\wedge\rightarrow$E may not correspond to the reader's expectations: That is, one might have expected the following: "*from* $(A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ | t_1,o_1,r_1, $(A_1\wedge\ldots\wedge A_n)$ | t_2,o_2,r_2 *and Combine*$((A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m),(A_1\wedge\ldots\wedge A_n))$ *infer* $(C_1\wedge\ldots\wedge C_m)$ | $\Lambda(t_1,t_2)$,$o_1\cup o_2$, $\mu\{r_1,r_2\},\{o_1,o_2\}$)". The reason why this is not so is related to the fact that the wff $(A_1,A_2,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ is considered to be an abbreviation of the wff $(A_1\rightarrow(A_2\rightarrow\ldots\rightarrow(A_n\rightarrow(C_1\wedge\ldots\wedge C_m))))$[32] (in which the order of the antecedents is irrelevant) as opposed to an abbreviation of $(A_1\wedge\ldots\wedge A_n)\rightarrow(C_1\wedge\ldots\wedge C_m)$. What the $\wedge\rightarrow$E rule does, in fact, is to apply the rule of MP n times, and what the rule of $\wedge\rightarrow$I does is also to apply SWM's $\rightarrow$I n times. A close look at the rule of $\rightarrow$I will show that if one would want $(A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ to be a syntactic abbreviation to $(A_1\wedge\ldots\wedge A_n)\rightarrow(C_1\wedge\ldots\wedge C_m)$, then the rule of $\wedge\rightarrow$I would have to be stated as follows: "from $(C_1\wedge\ldots\wedge C_m)$ | der,o$\cup\{A_1\wedge\ldots\wedge A_n\}$,r infer $(A_1,\ldots,A_n)\wedge\rightarrow(C_1,\ldots,C_m)$ | der,o,r-(o-$\{A_1\wedge\ldots\wedge A_n\}$)".

---

[32] This way of looking at $\wedge\rightarrow$ as an abbreviation of n nested entailments enables the implementation of some strategies to increase the efficiency of the inference system - refer to [Martins 83].

5.2. Or-Entailment

The or-entailment is another generalization of SWM's entailment to take a set of antecedents and a set of consequents. Or-entailment, written $\vee\!\!\rightarrow$, takes as arguments two sets of propositions. The proposition represented by the wff $(A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m)$ asserts that any antecedent relevantly implies the conjunction of the consequents. In other words, $(A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m)$ can be thought of as a shorthand for $A_1 \rightarrow (C_1 \wedge \ldots \wedge C_m) \wedge \ldots \wedge A_n \rightarrow (C_1 \wedge \ldots \wedge C_m)$.

Formally, within SWM, the rules for or-entailment introduction and elimination can be stated as follows:

Or-Entailment Introduction ($\vee\!\!\rightarrow$I): If for each k such that $1 \leq k \leq n$ we have $(C_1 \wedge \ldots \wedge C_m)$ $\mathrm{der}_{\mathcal{O}\cup\{A_k\},\mathcal{F}_k}$,[33] infer $(A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m) \mid \mathrm{der}_{\mathcal{O},\mathcal{F}}(o)$.

Or-Entailment Elimination ($\rightarrow\vee$E): From $(A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m) \mid t_1, o_1, r_1$, A t_2, o_2, r_2, $A \in \{A_1, \ldots, A_n\}$ and Combine$(A, (A_1, \ldots, A_n) \vee\!\!\rightarrow (C_1, \ldots, C_m))$, infer $C_1 \wedge \ldots \wedge C_m \mid A(t_1, t_2), o_1 \cup o_2, \mu(\{o_1, o_2\}, \{r_1, r_2\})$.

5.3. And-Or

And-or is a connective that generalizes $\neg$, $\wedge$, $\vee$,[34] $\oplus$ (exclusive or), $\mid$ (nand), and $\downarrow$ (nor). And-or, written ${}_n\mathbb{X}_i^j$, takes as arguments a set of n propositions. The proposition represented by the wff ${}_n\mathbb{X}_i^j(P_1, \ldots, P_n)$ asserts that there is a relevant connection between the propositions represented by the wffs $P_1, \ldots, P_n$ such that at least i and at most j of them must simultaneously be true. In other words, if n−i arguments of ${}_n\mathbb{X}_i^j$ are false, then the remaining i have to be true, and if j arguments of ${}_n\mathbb{X}_i^j$ are true then the remaining n−j have to be false. In and-or any argument can either be in consequent or antecedent position — i.e., used to

[33]This means that for each such k, the entailment $A_k \rightarrow (C_1 \wedge \ldots \wedge C_m) \mid \mathrm{der}_{\mathcal{O},\mathcal{F}}(o)$ holds, and therefore from the n previous wffs we can assert $(A_1 \rightarrow (C_1 \wedge \ldots \wedge C_m) \wedge \ldots \wedge A_n \rightarrow (C_1, \ldots, C_m)) \mid \mathrm{der}_{\mathcal{O},\mathcal{F}}(o)$

[34]There is another non-standard connective, truth functional and-or which generalizes truth functional or. This will not be discussed in this paper; it is presented in [Martins 83].

supply information to deduce the other arguments, or being deduced form the information gathered about the other arguments.

We will now define, without considering the information contained in the OT, OS and RS, what is needed to introduce and-or and what inferences can be drawn from it. Afterwards we will formally express the introduction and elimination rules within the SWM formalism.

Suppose that one wants to introduce ${}_n\mathbb{X}_i^j(P_1, \ldots, P_n)$: the rule of $\mathbb{X}$I requires the verification of the following two sets of conditions:

1. If any n−i arguments are false, then all the others have to be true. This can be stated as: $\forall k : 1 \leq k \leq \binom{n}{n-i} \wedge[\neg < {}_k c_{n-i}^n >] \rightarrow \wedge[{}_k \bar{c}_{n-i}^n]$

2. If any j arguments are true, then all the others have to be false. This can be stated as: $\forall k : 1 \leq k \leq \binom{n}{j} \wedge[{}_k c_j^n] \rightarrow \wedge[\neg < {}_k \bar{c}_j^n >]$.

Therefore, given n, i, and j, the $\mathbb{X}$I rule requires the verification of the $\binom{n}{n-i}$ entailments listed under 1 above and of the $\binom{n}{j}$ entailments listed under 2 above.

Informally, the following are the inferences allowed by $\mathbb{X}$E:

1. From ${}_n\mathbb{X}_i^j(P_1, \ldots, P_n)$, $\neg A_1$, ..., and $\neg A_{n-i}$, in which $\{A_1, \ldots, A_{n-i}\} \subset \{P_1, \ldots, P_n\}$, infer $\wedge[\{P_1, \ldots, P_n\} - \{A_1, \ldots, A_{n-i}\}]$.

2. From ${}_n\mathbb{X}_i^j(P_1, \ldots, P_n)$, A_1, ..., and A_j, in which $\{A_1, \ldots, A_j\} \subset \{P_1, \ldots, P_n\}$, infer $\wedge[\neg < \{P_1, \ldots, P_n\} - \{A_1, \ldots, A_j\} >]$.

Formally, within SWM, the rules for introducing and eliminating and-or can be stated as follows:

And-Or Introduction ($\mathbb{X}$I): If for each k such that $1 \leq k \leq \binom{n}{n-i}$, we have $A_1 \wedge \ldots \wedge A_i \mid \text{der}, o_k \cup \{\neg B_1, \ldots, \neg B_{n-i}\}, r_k$, in which $\{A_1, \ldots, A_i\} = {}_k \bar{c}_{n-i}^n$, and $\{B_1, \ldots, B_{n-i}\} = {}_k c_{n-i}^n$, meaning that $(\neg B_1, \ldots, \neg B_{n-i}) \wedge \neg(A_1, \ldots, A_i) \mid \text{der}, o_k, (o_k)$ and thereby that the $\binom{n}{n-i}$ entailments listed under 1 above are verified (we will refer to

each of these supported wffs as $F_k \mid der,O_k,R_k$); and also if for each q such that $1 \leq q \leq \binom{n}{j}$, we have $\neg A_1 \wedge \ldots \wedge \neg A_{n-j} \mid der,o'_q \cup \{B_1, \ldots, B_j\},r'_q$ in which $\{A_1, \ldots, A_{n-j}\} = {}_q\bar{c}^n_j$ and $\{B_1, \ldots, B_j\} = {}_qc^n_j$, meaning that $(B_1, \ldots, B_j) \wedge \neg(\neg A_1, \ldots, \neg A_{n-j}) \mid der,o'_{q,f} \cup o'_q$ and thereby that the $\binom{n}{j}$ entailments listed under 2 above are verified (we will refer to each of these supported wffs as $G_q \mid der,O'_q,R'_q$), then, letting $O=\{O_1, \ldots, O_{\binom{n}{n-i}}, O'_1, \ldots, O'_{\binom{n}{j}}\}$ and $R=\{R_1, \ldots, R_{\binom{n}{n-i}}, R'_1, \ldots, R'_{\binom{n}{j}}\}$:

1) If $\forall \alpha,\beta \in O$, $\alpha=\beta$ infer ${}_n\mathbb{X}^j_i(P_1, \ldots, P_n) \mid der,O_1,R_1$;[35]

2) If $\exists \alpha,\beta \in O$ such that $\alpha \neq \beta$ and also if Combine$(F_1, \ldots, F_{\binom{n}{n-i}}, G_1, \ldots, G_{\binom{n}{j}})$, infer ${}_n\mathbb{X}^j_i P_1, \ldots, P_n) \mid ext,\cup\{O\},\mu(R,O)$.

And-Or Elimination ($\mathbb{X}$E): From ${}_n\mathbb{X}^j_i(P_1, \ldots, P_n) \mid t,o,r, \neg A_1 \mid t_1,o_1,r_1, \ldots, \neg A_{n-i} \mid t_{n-i},o_{n-i},r_{n-i}, \{A_1, \ldots, A_{n-i}\} \subset \{P_1, \ldots, P_n\}$, and Combine$({}_n\mathbb{X}^j_i(P_1, \ldots, P_n), \neg A_1, \ldots, \neg A_{n-i})$, then, denoting by $\{B_1, \ldots, B_i\}$ the set $\{P_1, \ldots, P_n\} - \{A_1, \ldots, A_{n-i}\}$ and by O and R the sets $O=\{o,o_1, \ldots, o_{n-i}\}$ and $R=\{r,r_1, \ldots, r_{n-i}\}$, infer $B_1 \wedge \ldots \wedge B_i \mid \Lambda(t,t_1, \ldots, t_{n-i}),\cup\{O\},\mu(R,O)$;

From ${}_n\mathbb{X}^j_i(P_1, \ldots, P_n) \mid t,o,r, A_1 \mid t_1,o_1,r_1, \ldots, A_j \mid t_j,o_j,r_j, \{A_1, \ldots, A_j\} \subset \{P_1, \ldots, P_n\}$ and Combine$({}_n\mathbb{X}^j_i(P_1, \ldots, P_n), A_1, \ldots, A_j)$, then, denoting by $\{B_1, \ldots, B_{n-j}\}$ the set $\{P_1, \ldots, P_n\} - \{A_1, \ldots, A_j\}$ and by O and R the sets $O=\{o,o_1, \ldots, o_j\}$ and $R=\{r,r_1, \ldots, r_j\}$, infer $\neg B_1 \wedge \ldots \wedge \neg B_{n-j} \mid \Lambda(t,t_1, \ldots, t_j),\cup\{O\},\mu(R,O)$.

As a last point in the discussion of $\mathbb{X}$I, let us consider the cases in which i=0, j=n, i=j=0, and i=j=n. When trying to introduce $\mathbb{X}$ in any of these cases, we are faced with entailments that have either empty antecedent or empty consequent. When this happens, the proof of the entailments with empty antecedent or consequent should be disregarded and only the other set of entailments should be considered. Notice that we are not saying anything about how to introduce entailments with empty consequent or empty antecedent; in fact, such formulas are

[35]Notice that since all the O's are equal then all the R's are equal as well (Theorem 5, Appendix 1) and for that reason, it doesn't matter which O's or R's we write down in the final wff.

not wffs, and thus we don't have to worry about them. What we are saying here is that for some values of i and j, the ⋇ connective becomes somewhat simplified, and thus not so much work is required in its introduction.

5.4. Thresh

Thresh generalizes equivalence to a set of arguments. Thresh, written ${}_n\Theta_i$, takes as arguments a set of n propositions. The proposition represented by the wff ${}_n\Theta_i(P_1, \ldots, P_n)$ asserts that there is a relevant connection between the propositions represented by the wffs $P_1, \ldots, P_n$ such that either fewer than i of them are true or they all are true. If at least i of the arguments of ${}_n\Theta_i$ are true then all the remaining arguments have to be true and if i−1 arguments of ${}_n\Theta_i$ are true and at least one is false, then the remaining arguments have to be false. As in the last section, these inferences will be the guidelines for stating the ${}_n\Theta_i$I rule and will be explicitly stated by the ${}_n\Theta_i$E rule.

The introduction of ${}_n\Theta_i(P_1, \ldots, P_n)$ requires the verification of the following two sets of conditions:

1. $\forall k : 1 \leq k \leq \binom{n}{i} \wedge [{}_k C_i^n] \rightarrow \wedge [{}_k \bar{C}_i^n]$
2. $\forall k : 1 \leq k \leq \binom{n}{i} \; \forall P \in {}_k C_i^n : \wedge [{}_k C_i^n - \{P\} \cup \{\neg P\}] \rightarrow \wedge [\neg {}_k \bar{C}_i^n]$

Given n and i, the ${}_n\Theta_i$I rule requires the verification of $\binom{n}{i}$ conditions of type 1 and $i \times \binom{n}{i}$ conditions of type 2 (there are $\binom{n}{i}$ different ways of choosing the antecedents of the entailment, and for each of those ways there are i ways of choosing the argument that is negated).

The conditions for the elimination rule are obtained directly from the conditions of the ${}_n\Theta_i$I rule:

1. From ${}_n\Theta_i(P_1, \ldots, P_n)$, $A_1, \ldots, A_i$, and $\{A_1, \ldots, A_i\} \subset \{P_1, \ldots, P_n\}$, infer $B_1 \wedge \ldots \wedge B_{n-i}$, where $\{B_1, \ldots, B_{n-i}\} = \{P_1, \ldots, P_n\} - \{A_1, \ldots, A_i\}$.
2. From ${}_n\Theta_i(P_1, \ldots, P_n)$, $A_1, \ldots, A_{i-1}$, and $\neg A_i$, where $\neg A_i \notin \{A_1, \ldots, A_{i-1}\}$, and $\{A_1, \ldots, A_{i-1}, A_i\} \subset \{P_1, \ldots, P_n\}$, infer $\neg B_1 \wedge \ldots \wedge \neg B_{n-i}$, where $\{B_1, \ldots, B_{n-i}\} = \{P_1, \ldots$

$\ldots, P_n\}-\{A_1, \ldots, A_i\}$.

Formally, within SWM, the rules for thresh introduction and elimination can be stated as follows:

Thresh Introduction (ΘI): If for each k such that $1 \leq k \leq \binom{n}{i}$ we have $A_1 \wedge \ldots \wedge A_{n-i} \mid \text{der}, o_k \cup \{B_1, \ldots, B_i\}, r_k$, **where** $\{A_1, \ldots, A_{n-i}\} = {}_k\bar{c}_i^n$, and $\{B_1, \ldots, B_i\} = {}_k c_i^n$, meaning that $(B_1, \ldots, B_i) \wedge \neg(A_1, \ldots, A_{n-i}) \mid \text{der}, o_k, (o_k)$ (which we will refer to as $F_k \mid \text{der}, O_k, R_k$),[36] and also if for each q such that $1 \leq q \leq \binom{n}{i}$ and for each E such that $E \in {}_q c_i^n$ we have $\neg A_1 \wedge \ldots \wedge \neg A_{n-i} \mid \text{der}, o'_q \cup (\{B_1, \ldots, B_i\} - \{E\}) \cup \{\neg E\}, r'_q$, where $\{A_1, \ldots, A_{n-i}\} = {}_q\bar{c}_i^n$, and $\{B_1, \ldots B_i\} = {}_q c_i^n$, meaning that $((\{B_1, \ldots, B_i\} - \{E\}) \cup \{\neg E\}) \wedge \neg(\neg A_1, \ldots, \neg A_{n-i}) \mid \text{der}, o'_q, (o'_q)$ (we will refer to these supported wffs as $G_s \mid \text{der}, o'_s, R'_s$;[37] then, letting $O = \{O_1, \ldots, O_{\binom{n}{i}}, O'_1, \ldots, O'_{\binom{n}{i}}\}$ and $R = \{R_1, \ldots, R_{\binom{n}{i}}, R'_1, \ldots, R'_{\binom{n}{i}}\}$:

1) If $\forall \alpha, \beta \in O$, $\alpha = \beta$, infer ${}_n\Theta_i(P_1, \ldots, P_n) \mid \text{der}, O_1, R_1$;

2) If $\exists \alpha, \beta \in O$ such that $\alpha \neq \beta$, and Combine$(F_1, \ldots, F_{\binom{j}{i}}, G_1, \ldots, G_{i \times \binom{n}{i}})$, infer ${}_n\Theta_i(P_1, \ldots, P_n) \mid \text{ext}, \cup\{O\}, \mu(R, O)$.

Thresh Elimination (ΘE):

1) If ${}_n\Theta_i(P_1, \ldots, P_n) \mid t, o, r$, $A_1 \mid t_1, o_1, r_1$, $\ldots$ and $A_i \mid t_i, o_i, r_i$ are supported wffs, $\{A_1, \ldots, A_i\} \subset \{P_1, \ldots, P_n\}$, and Combine$({}_n\Theta_i(P_1, \ldots, P_n), A_1, \ldots, A_i)$, then, denoting by O and R the sets $O = \{o, o_1, \ldots, o_i\}$ and $R = \{r, r_1, \ldots, r_i\}$, infer $B_1 \wedge \ldots \wedge B_{n-i} \mid \Lambda(t, t_1, \ldots, t_i), \cup\{O\}, \mu(R, O)$ in which $\{B_1, \ldots, B_{n-i}\} = \{P_1, \ldots, P_n\} - \{A_1, \ldots, A_i\}$.

2) If ${}_n\Theta_i(P_1, \ldots, P_n) \mid t, o, r$, $A_1 \mid t_1, o_1, r_1$, $\ldots$, $A_{i-1} \mid t_{i-1}, o_{i-1}, r_{i-1}$ and $\neg A_i \mid t_i, o_i, r_i$ are supported wffs, $\{A_i\} \cap \{A_1, \ldots, A_{i-1}\} = \emptyset$, $\{A_1, \ldots A_{i-1}, A_i\} \subset \{P_1, \ldots,$

---

[36] This means that the $\binom{n}{i}$ entailments listed under 1 are verified.

[37] Notice that for each q there are i of these supported wffs, i.e., s ranges from 1 to $i \times \binom{j}{i}$, and thereby that the $i \times \binom{j}{i}$ entailments listed under 2 are verified.

$P_n\}$ and Combine($_n\mathcal{O}_i(P_1, \ldots, P_n), A_1, \ldots, A_{i-1}, \neg A_i$) then, denoting by O and R the sets $O=\{o, o_1, \ldots, o_{i-1}, o_i\}$ and $R=\{r, r_1, \ldots, r_{i-1}, r_i\}$, infer $\neg B_1 \wedge \ldots \wedge \neg B_{n-i} \mid \Lambda(t, t_1, \ldots, t_{i-1}, t_i), \cup\{O\}, \mu(R, O)$ in which $\{B_1, \ldots, B_{n-i}\}=\{P_1, \ldots, P_n\}-\{A_1, \ldots, A_i\}$.

6. MBR - The Abstract Level of a Belief Revision System

Having presented the SWM system, we now discuss how a belief revision system using SWM should interpret SWM's wffs and how SWM's features can be used in applications of belief revision. In this section, we provide what we call a *contextual interpretation* for SWM. We use the word "contextual interpretation" instead of just "interpretation" for the following two reasons: On the one hand, we want to stress that we are not providing an interpretation for SWM in the logician's sense of the word; on the other hand, we want to emphasize that our definition of truth depends on the notion of context. This contextual interpretation defines the behavior of an abstract assumption-based belief revision system (i.e., not tied to any particular implementation), which we call MBR (**M**ultiple **B**elief **R**easoner).

We will assume that MBR works with a knowledge base containing propositions that are associated with an OT, OS, and RS (in SWM's sense). The propositions that are added to the knowledge base follow the rules of inference of SWM.

We define a *context* to be a set of hypotheses. A context determines a *Belief Space* (BS), which is the set of all the hypotheses defining the context and all the propositions that were derived exclusively from them. Within the SWM formalism, the wffs in a given BS are characterized by having an OS that is contained in the context. The set of contexts represented in the knowledge base is the power set of the set of hypotheses existing in the knowledge base.

Any operation performed within the knowledge base (query, addition, deletion, etc.) will be associated with a context. We will refer to the context under consideration, i.e., the context associated with the operation currently being performed in the knowledge base, as the *current context*. While the operation is being carried out, the only propositions that will be considered

are the propositions in the BS defined by the current context. This BS will be called the *current belief space*. A proposition is said to be *believed* if it belongs to the current BS. We can look at contexts as delimiting smaller knowledge bases (namely, the Belief Spaces) within the knowledge base. The only propositions in the knowledge base that are retrievable are those propositions that belong to the current BS.

A common goal of belief revision systems is to stay away from contradictions, i.e., to avoid the simultaneous belief of a proposition and its negation. Taking this into account, it would seem natural to constrain contexts to be consistent sets of hypotheses, not just any sets of hypotheses. Let us note, however, that determining whether a contradiction is derivable from a set of hypotheses is a difficult problem in logic, and thus the condition that contexts are not inconsistent may be very difficult to enforce. For that reason, we may settle for the weaker condition that contexts are not known to be inconsistent.

Within MBR, we can easily detect whether a context is not known to be inconsistent by considering the RSs of the hypotheses defining the context. Given the context $\{H_1, \ldots, H_n\}$, the condition

$$\forall H \in \{H_1, \ldots, H_n\}\ \forall r \in \mathbf{rs}(H) : r \subset \{H_1, \ldots, H_n\} - \{H\}$$

guarantees that the context $\{H_1, \ldots, H_n\}$ is not known to be inconsistent.[38]

However, it may be the case that in MBR one desires to perform reasoning within the BS defined by an inconsistent context (a kind of counterfactual reasoning). In SWM, the existence of contradictions is not as damaging as in classical logic, in which anything can be derived from a contradiction. Thus, in MBR one may not want to bother discarding hypotheses after a contradiction is detected, since the contradiction will not affect the entire system.

For these reasons, in MBR, the condition that a context is not known to be inconsistent will not be compulsory but rather advisable if one doesn't explicitly want to perform

[38] The condition $\exists H \in \{H_1, \ldots, H_n\}\ \exists r \in \mathbf{rs}(H) : r \subset \{H_1, \ldots, H_n\} - \{H\}$ guarantees that the context $\{H_1, \ldots, H_n\}$ is known to be inconsistent.

reasoning in a BS that is known to be inconsistent. The reason why it is advisable is that within a BS defined by a context not known to be inconsistent some simplification can be considered during the application of the rules of inference, as stated by the following theorems (their proof can be found in the Appendix 1):

Theorem 6: If C is a context that is not known to be inconsistent, then, for any two wffs, A and B, in the BS defined by the context C, we have Combine(A,B)=true.

Corollary 6.1: If one uses a context which is not known to be inconsistent, then MBR does not need to check for combinability between the wffs before the application of rules of inference.

Let us now consider how MBR acts when a contradiction is detected. We will discuss two levels of belief revision: belief revision within the current context and belief revision within a context strictly containing the current context. The main difference between them is that the former may require changes in the current context and allows the deduction of new wffs, while the latter leaves this context unchanged and does not allow the deduction of new wffs to the knowledge base.

SWM has two rules of inference to handle contradictions: negation introduction ($\neg$I) and updating of restriction sets (URS).

The rule of $\neg$I states that from the combinable supported wffs $A \mid t_1,o_1,r_1$ and $\neg A \mid t_2,o_2,r_2$, we can deduce the negation of the conjunction of any number of hypotheses in $o_1 \cup o_2$ under an OS containing the remaining hypotheses. This rule is applied whenever two contradictory wffs are found within the current BS. Its effect is twofold: (1) It may cause the current context to be changed. The fact that both A and $\neg$A were derived within the current BS means that the current context is *now known* to be an inconsistent set. If one wants to maintain contexts that are not known to be inconsistent, then the current context has to be changed. (2) It allows the deduction of new wffs to the knowledge base. Such wffs are negations of conjunctions, whose conjuncts are some hypotheses in the current context (the

hypotheses in $o_1 \cup o_2$).

The rule of URS has the effect of recording the occurrence of contradictions in the RSs of all the hypotheses underlying a contradiction (and the wffs derived from them). This rule, however, does not allow the addition of new wffs to the knowledge base. This rule is obligatorily applied whenever two contradictory wffs are found, whether or not they belong to the current BS. Upon application of this rule, there will be an explicit record in the knowledge base about the possibility of the derivation of the contradictory wffs.

When a contradiction is detected, one of two things will happen:

1. *Only one of the contradictory wffs belongs to the current BS*:[39] the contradiction is recorded (through the application of URS), but nothing more happens. The effect of doing so is to record that some set of hypotheses, properly containing the current context, is now known to be inconsistent. This results in what we call *belief revision within a context properly containing the current context*. This type of revision of beliefs has the effect of recording that a BS larger than the current BS is inconsistent.

2. *Both contradictory wffs belong to the current BS*: URS is applied, resulting in the updating of the RSs of the hypotheses in the current context (and the derived wffs in the current BS), and, in addition, the rule of $\neg$I may also be applied. This results in what we call *belief revision within the current context*, normally originating the disbelief (removal from the current context) of some of the hypotheses in the current context.

7. SNeBR - A SNePS Implementation of MBR

In this section we describe a particular implementation of MBR using the SNePS semantic network processing system [Shapiro 79a]. The system we describe is called SNeBR.[40] The aspects of SNeBR discussed in this section are: the representation of propositions, in particular,

[39]Note that at least one of the contradictory wffs belongs to the current BS, since a contradiction is detected whenever some newly derived wff contradicts some existing one, and newly derived wffs always belong to the current BS.

[40]SNePS Belief Revision

the representation of contradictory propositions and the representation of propositions obtained by multiple derivations; the representation of contexts; the mechanism used by the network matching function in order to retrieve only the propositions in the BS under consideration; the basic functions available to SNeBR users, briefly introducing the SNeBR inference system; and, finally, some details of the implementation of the process of revision of beliefs.

When using SNeBR, one can perform the following operations:

1. Add new hypotheses to the network. There exists a function that allows a user to specify a proposition and the name of a context; the proposition is added to the network, justified as a hypothesis, and added to the hypotheses that constitute the named context.[41]

2. Name a context. One can assign a name to a given context and use that name whenever the context is being referenced, rather than listing every hypothesis in the context.

3. Ask for all the nodes in a given BS that match a given pattern. One can specify a node (which may contain free variables) and a context; the network matching function will retrieve all the nodes that match the specified pattern and are part of the BS defined by the context.

4. Perform backward inference in the BS defined by a given context. One can ask SNeBR to deduce a given proposition (possibly containing free variables) in the BS defined by a context. SNeBR will retrieve relevant deduction rules in the specified BS and will create a set of processes to derive the desired instances.

5. Perform forward inference in the BS defined by a given context. One can also ask SNeBR to perform forward reasoning with a given hypothesis in a given context. In this case, the hypothesis is built into the network, added to the context under consideration, and a set of processes is built to find all the consequences of the added hypothesis.

[41] In this function, and in all the other user functions available in SNeBR, if no context name is specified, it defaults to the name "current context".

A SNePS semantic network [Shapiro 79a] is a labeled directed graph in which nodes represent intensional concepts and arcs represent non-conceptual binary relations between concepts. One of the assumptions underlying the SNePS network is the so called *uniqueness principle* [Maida and Shapiro 82], which states that each concept is represented in the network by a *unique* node. An arc is labeled with a symbol intended to be mnemonically suggestive of the relation the arc represents. The relations represented solely by arc labels are not conceptual: they are used to form the basic structure of the semantic network. Whenever an arc representing a relation **r** goes from node **n** to node **m**, there is an arc representing the converse relation of **r**, $\mathbf{r}^{c}$, going from **m** to **n**.

In SNeBR, propositions are represented by SNePS nodes. Associated with each node representing a proposition, there is another node, representing its support (called the *supporting node*). The supporting node has arcs labeled **os** that point to the nodes representing the hypotheses in the OS of the proposition and arcs labeled **rs** that point to the sets in the restriction set of the proposition. Each of these sets is, in turn, represented by a node that has arcs labeled **ers** (element of the restriction set) to each hypothesis that it contains. The OT represents a relation between a proposition and its support and is represented in the network by an arc (labeled either **hyp**, **der** or **ext**)[42] that connects the node representing the proposition with the node representing its support. In Figures 4, 5, and 6, we show the network representation of hypotheses and derived propositions (propositions with "der" or "ext" OTs).

[42]We refer to these arcs as OT, and capitalize the label in the arc name to stress that the label stands for an abbreviation of either **hyp**, **der** or **ext**.

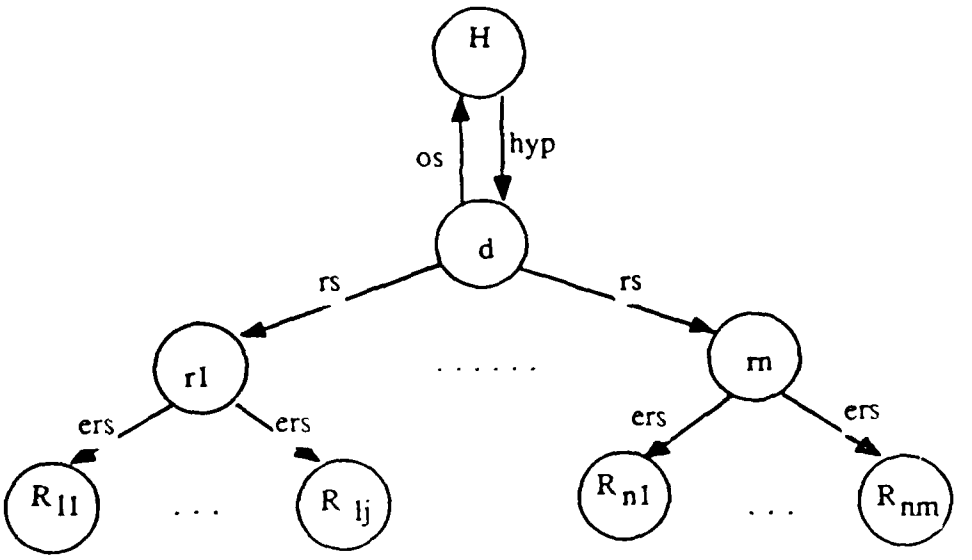


Figure 4
representation of H | hyp,{H},{{R_{11} . . . R_{1j}}. . . {R_{n1} . . . R_{nm}}}

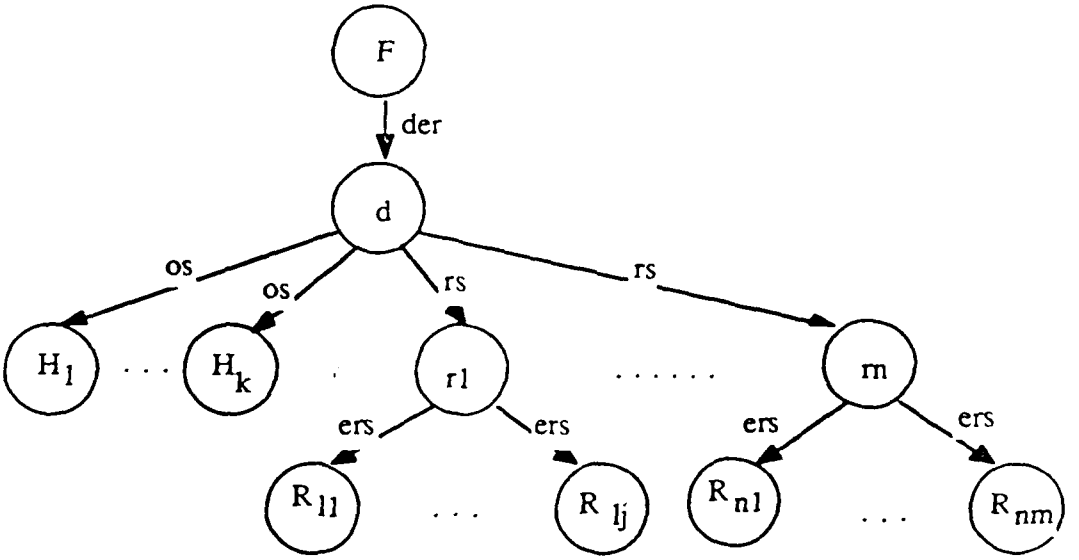


Figure 5
representation of F | der,{H_1 . . . H_k},{{R_{11} . . . R_{1j}}. . . {R_{n1} . . . R_{nm}}}

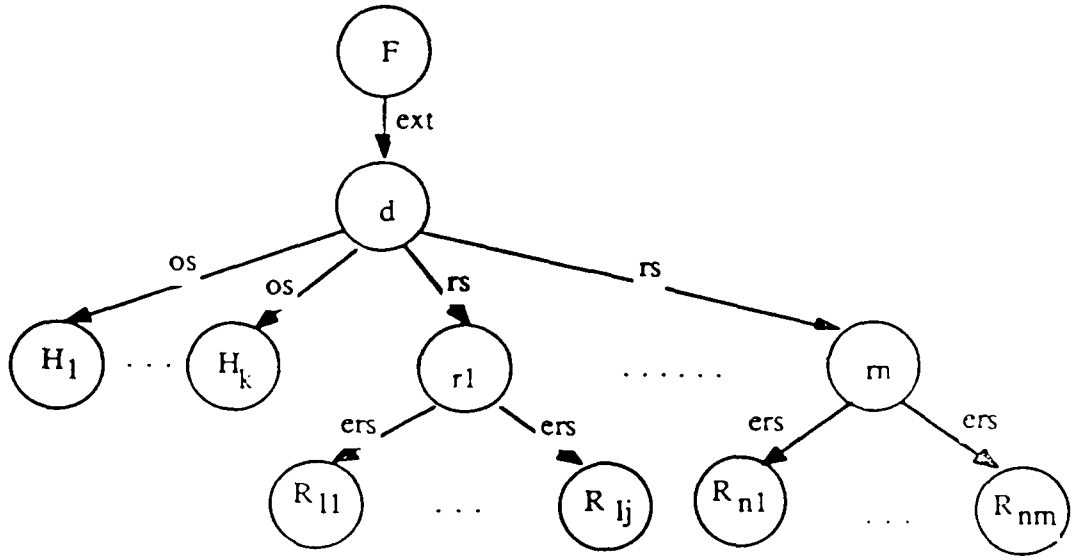


Figure 6
representation of $F \mid ext,\{H_1 \ldots H_k\},\{\{R_{11} \ldots R_{1j}\} \ldots \{R_{n1} \ldots R_{nm}\}\}$

Let us now describe how contradictory propositions are represented in the network. The uniqueness principle guarantees that there is as much sharing among network structures as possible. According to this principle, two nodes representing contradictory wffs should share some common network structure. To illustrate this aspect about network sharing, let us consider, for example, the supported wffs $P \mid t_1,\{H_1,H_2\},\{\}$ and $\neg P \mid t_2,\{H_3,H_4\},\{\}$. These supported wffs have in common the proposition P. Their network representation is shown in Figure 7.[43] In this figure the node n5 represents the proposition P and the node n6 represents the proposition $\neg P$.[44]

[43]In this figure and in the forthcoming figures, a triangle hanging from a node denotes the network structure necessary to represent the proposition that is written below the triangle.

[44]Notice that $\neg P$ is represented by ${}_1\mathcal{W}_0^0(P)$. The proposition ${}_n\mathcal{W}_i^j(P_1, \ldots, P_n)$ is represented in the network by a node with arcs labeled **arg** to the nodes representing $P_1, \ldots, P_n$, an arc labeled **min** to i and an arc labeled **max** to j.

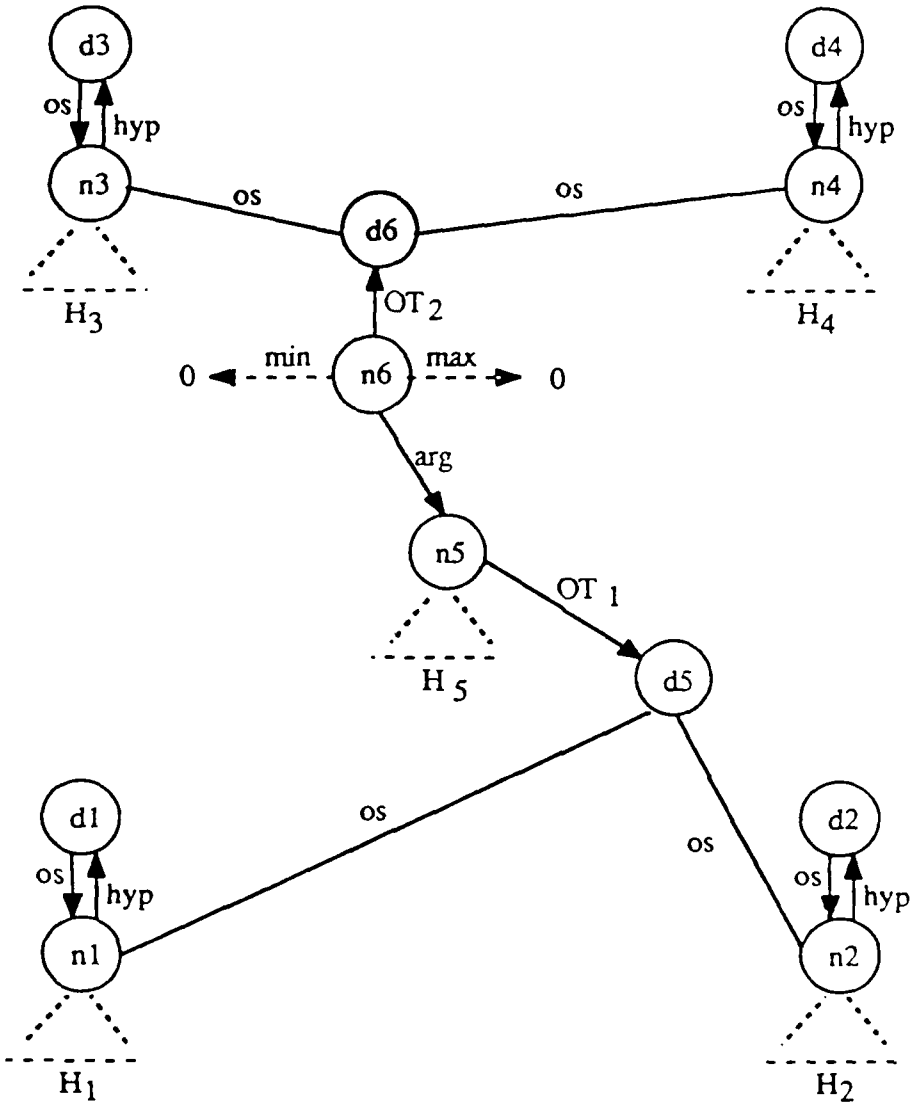


Figure 7
Representation of contradictory propositions

To obtain full sharing of network structures when a new node is about to be built, SNeBR first checks whether that node (or its negation) already exists in the network; in this case, the entire network is considered, not just that portion of it that belongs to the BS under consideration. The network matching function [Shapiro 77], [Saks 85] *guarantees* that this

node is found without having to search every node in the network.

We also need to consider the problem of multiple derivations of the same proposition. The reason that the issue of multiple derivations of the same proposition is raised here rather than in Section 4 is that this is a representational issue rather than a logical one. Within SWM, if the same wff is derived twice from different sets of hypotheses, then it is written twice in the knowledge base with the corresponding OT, OS, and RS. When dealing with issues of representation, we have to decide how to represent those supported wffs, and thus the current discussion. Suppose that a given proposition is derived in several different ways (with different OSs and RSs); how should the multiple occurrences of the same proposition be represented? Should they share the same node? The uniqueness principle requires that the node representing this proposition be shared by the different occurrences of the proposition. Our decision in this case is to link the node that represents that proposition to multiple supporting nodes, each one of which represents one of the possible derivations. As an example, suppose that the network contained the nodes representing the supported wffs C | hyp,{C},{}; C | der,{A; A→C},{} and C | der,{B; B→C},{{D}}. The proposition C has three supporting nodes: one of them represents a hypothesis and the other two, represent derived propositions. The representation of the proposition C is depicted in Figure 8. In this figure, node d1 represents the support of C | hyp,{C},{}; node d2 represents the support of C | der,{A,A→C},{}; and node d3 represents the support of C | der,{B,B→C},{{D}}.

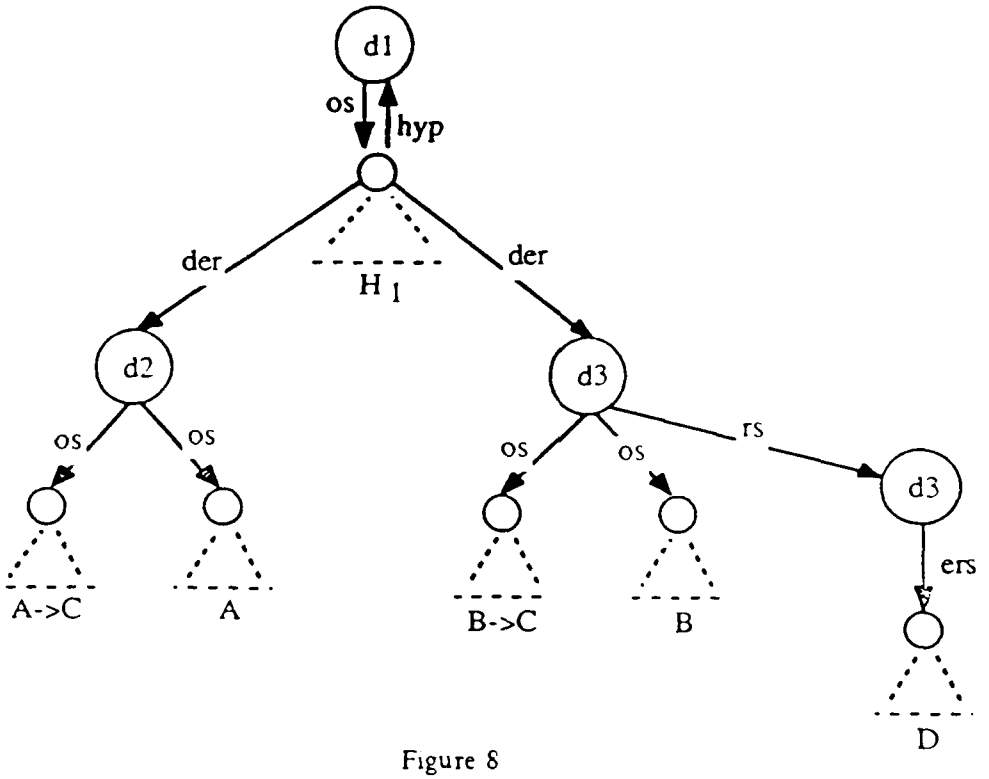


Figure 8
Representation of C

Notice furthermore that since supported wffs with the same OS have the same RS (Theorem 5, in Appendix 1), we allow network sharing between supporting nodes. This means that if there exist two nodes in the network representing propositions with the same OS, then those nodes will share a common supporting node. This decision, besides yielding some savings in memory, allows some savings in processing as well when a network update (URS) due to the detection of a contradiction is being performed.

In the network, a context is represented by a set of nodes, each one of which represents a hypothesis. In order to allow a SNeBR user to refer to and talk about contexts, contexts can be named. The name of a context is represented in the network by a node that has arcs labeled **:val** to each node defining the context. Figure 9 shows the network representation of a context

named "ct1" containing hypotheses H_1 and H_2.

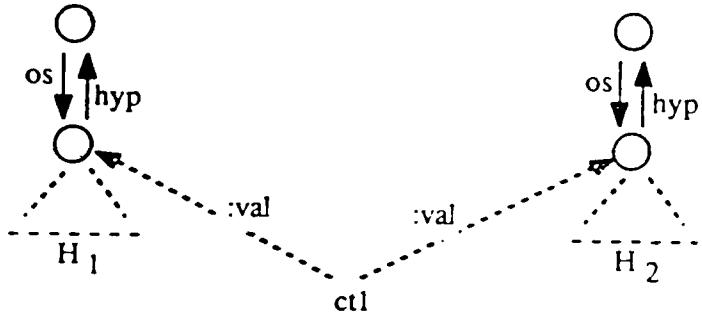


Figure 9
Representation of the context "ct1"

Based on the representation just presented, let us consider how the network matching function decides whether or not a given node should be retrieved from the network. Given a node and a context, the node belongs to the BS defined by the context if one of its supporting nodes has an OS contained in the context. Let us denote by **do**(n) the support[45] of the proposition represented by node n and by **os**(d) the OS represented by the supporting node d. Then, given the context $\{H_1, \ldots, H_n\}$, represented in the network by nodes $h_1, \ldots, h_n$, we say that node n belongs to the BS defined by this context if $\exists d \in do(n) : os(d) \subset \{h_1, \ldots, h_n\}$. The network matching function uses this condition to decide whether or not a given node should be retrieved. Suppose that the network matching function is given a node, say π, and a context Ω and discovers that node ψ in the network matches π.[46] To decide whether or not ψ should be retrieved, the network matching function computes **do**(ψ), which is done by following the **OT** arcs leaving ψ, and for each of the nodes in **do**(ψ) it checks whether the node's OS is contained in Ω. In other words, the network matching function does a pattern match as if there were no

[45]The *support* of a node is the set of its supporting nodes.

[46]Details of how this is done can be found in [Shapiro 77], [Martins 83] and [Saks 85]. For the purposes of this discussion it suffices to say that ψ is found without having to search every node in the network.

contexts and then "filters" the nodes in the BS under consideration. The question of whether this "filtering" can be done before the match without extensive computation is still an open problem.

The SNeBR inference system allows both backward and forward inference to be performed. Every deduction rule in the BS under consideration may be used, in either backward or forward inference or both. When a deduction rule is used it is activated and remains that way until explicitly de-activated by the user. The activated rules are assembled into a set of processes, called an *active connection graph* (acg) [McKay and Shapiro 80], which carry out the inferences. The acg also stores all the results generated by the activated rules. If during some deduction, the inference system needs some of the rules activated during a previous deduction, it uses their results directly instead of re-deriving them. Forward and backward inference interface smoothly, giving rise to a behavior that we call *bi-directional inference*. (For more details, refer to [Shapiro, Martins and McKay 82]).

There are two main concepts involved in the implementation of the inference system: pattern-matching and the use of procedural (or active) versions of deduction rules. The pattern-matching process is given a piece of the network (either to be deduced in backward inference or added in forward inference) and a context, and locates relevant deduction rules in the BS defined by the context. Such deduction rules are then "compiled" into a set of processes, which are given to a multi-processing system for execution.[47] The multi-processing system used by the inference system is called MULTI [McKay and Shapiro 80]; it is a LISP-based system mainly consisting of a simple evaluator, a scheduler, and system primitives. The evaluator continuously executes processes from a process queue until the queue becomes empty; the scheduler inserts processes into the process queue; system primitives include functions for creating processes, scheduling processes, and manipulating local variables or registers. Every

[47]The multi processing approach was influenced both by Kaplan's producer-consumer model [Kaplan 73] and by Wand's frame model of computation [Wand 74].

process has a name that defines the action the process will perform and also has a continuation link naming the process that is to be scheduled for activation after it has completed its job. Each process also has other "registers" peculiar to the action it will perform. Some processes receive answers and "remember" all the answers they received. Details of the inference system and the processes it uses can be found in [Martins 83].

In closing this section, we now describe how contradictions are handled by SNeBR. A contradiction will be detected by SNeBR when one of the following conditions occurs: (1) Nodes representing contradictory wffs are built into the BS under consideration;[48] (2) Information gathered by a connective elimination process shows that a rule is invalidated by the data in the BS. We will discuss each of these cases in turn.

Suppose that when the inference system builds a node (say n1) representing proposition P, it discovers that there is a node (say n2) representing the proposition ¬P (the discovery of n2 is guaranteed by the uniqueness principle) and suppose furthermore that n2 has a supporting node, meaning that *the proposition represented by n2 (¬P) belongs to some BS.*[49]

In this case, the rule of URS is immediately applied, having the effect of recording the new set known to be inconsistent. After that, the inference system investigates whether or not the current context has to be revised. This would happen if both n1 and n2 belong to the current BS.[50] If this is the case, then the rule of ¬I is applied and a decision has to be made about which hypothesis is the culprit for the contradiction. This decision is not made at the logical level (i.e., by the rules of inference) but rather at some other level. In the current implementation, this is done through an interaction with the user (see Section 8).

---

[48] If nodes representing contradictory propositions are built but one of them does not belong to the BS under consideration, SNeBR records that there is an inconsistent BS (which is not being considered) through the application of URS, and proceeds.

[49] If n2 does not have a supporting node, then it means that n2 is a component of another proposition and thus does not contradict n1. For example ¬P and PvQ are not contradictory propositions.

[50] The node n1 has just been derived; therefore it belongs to the current BS. The node n2 would belong to the current BS if one of its supporting nodes has an OS that is contained in the current context.

The other way to detect a contradiction in SNeBR is when a process trying to derive instances of the consequents of a deduction rule gathers information that invalidates the rule. Let us consider, for example, the deduction rule ${}_3\&_1^1(A,B,C)$ that states that *exactly* one of A, B, and C is true. If this deduction rule exists in the BS being considered and SNeBR is asked whether C is true it will try to find whether A or B are true. If it finds that *both* A and B are true then it reports a contradiction since the rule is invalidated by the data gathered. An example of this is shown in Section 8.

Let us now take a closer look at SNeBR's implementation of URS. This rule requires two traversals of the network: the first is to reach the hypotheses underlying the contradiction, updating their RSs; the second, in the other direction, is to update the RSs of all the wffs derived from them. When a contradiction is found, the computation of all hypotheses underlying it is done by following the **OT** and **os** arcs *directly* linking the contradictory wffs with the hypotheses that they assume. The updating of the RSs of the hypotheses consists of introducing a new set in the RS of each one of them (or in modifying some existing RS), which is done by creating nodes representing those sets (or by deleting some of the arcs in the existing sets). To update the wffs derived from those hypotheses, only one arc has to be traversed for each hypothesis (the $\mathbf{os}^C$ arc connecting the hypothesis with the supporting nodes of the wffs derived from it). Such wffs are updated by creating RSs or updating existing ones. This process can be better understood with some examples.

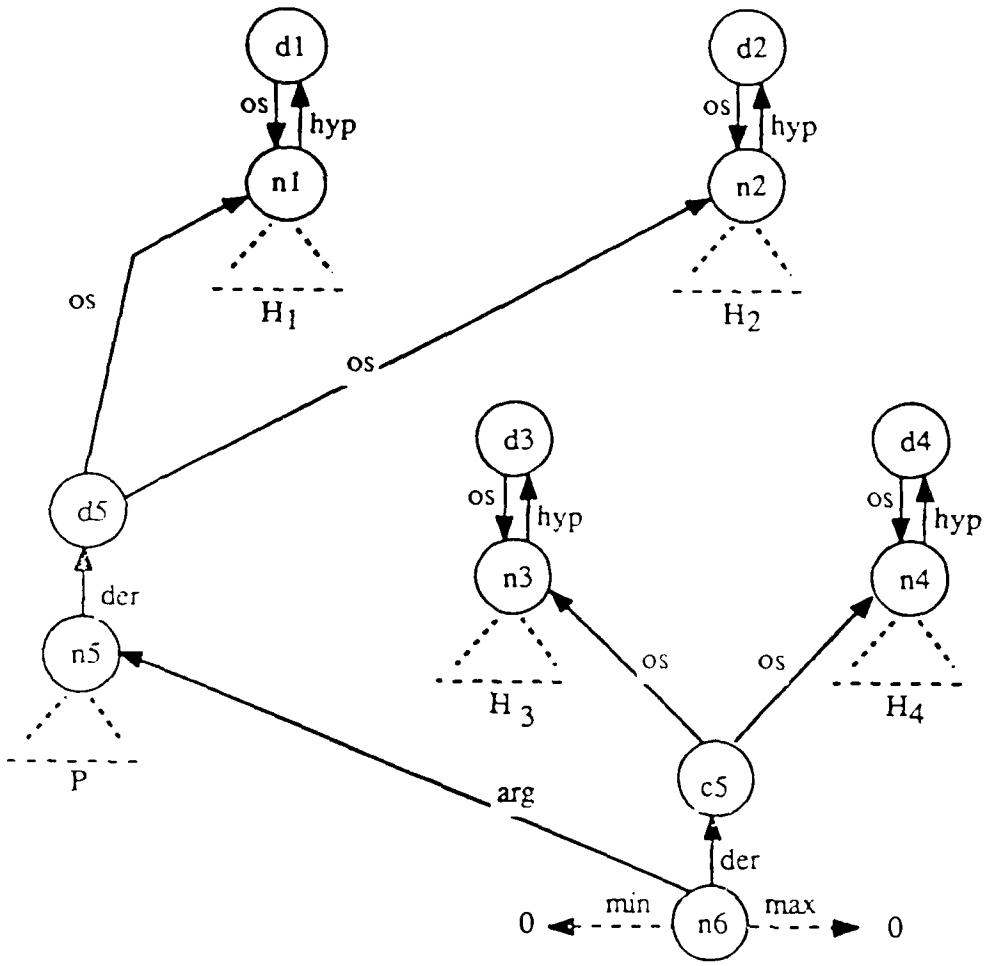


Figure 10
Network before URS

Consider the network represented in Figure 10 in which nodes n1, n2, n3, n4, and n5 represent, respectively, hypotheses H_1, H_2, H_3, H_4, and the derived proposition P. Suppose that ¬P, has just been derived. When SNeBR builds node n6 (representing ¬P), it detects a contradiction involving nodes n5 and n6, causing the rule of URS to be applied. The first step in the application of URS consists in computing the set of hypotheses underlying the contradiction. This set is computed by following the OT (in our example OT is der) and os arcs leaving the contradictory nodes: from n5 we obtain {n1, n2} and from n6 we obtain {n3, n4}; therefore,

the set {n1, n2, n3, n4} had just been discovered to be inconsistent. The next step consists in updating the RS of each of the hypotheses underlying the contradiction, which is done by creating a new RS for each hypothesis. For example, when updating the RS of n1, we create a new RS (r1 in Figure 11) with hypotheses represented by the nodes in the set {n2, n3, n4}. Finally, as the last step in the application of URS, the RSs of all the wffs (with der or ext OT) depending on the inconsistent set of hypotheses are updated. This is done by following the os^c arc leaving each of the contradictory hypotheses to find all the supporting nodes depending on that hypothesis (in this case, finding the supporting nodes d5 and d6) and updating their RS (in this case, the nodes r5 and r6 are created). Figure 11 represents the network of Figure 10 after the application of URS.

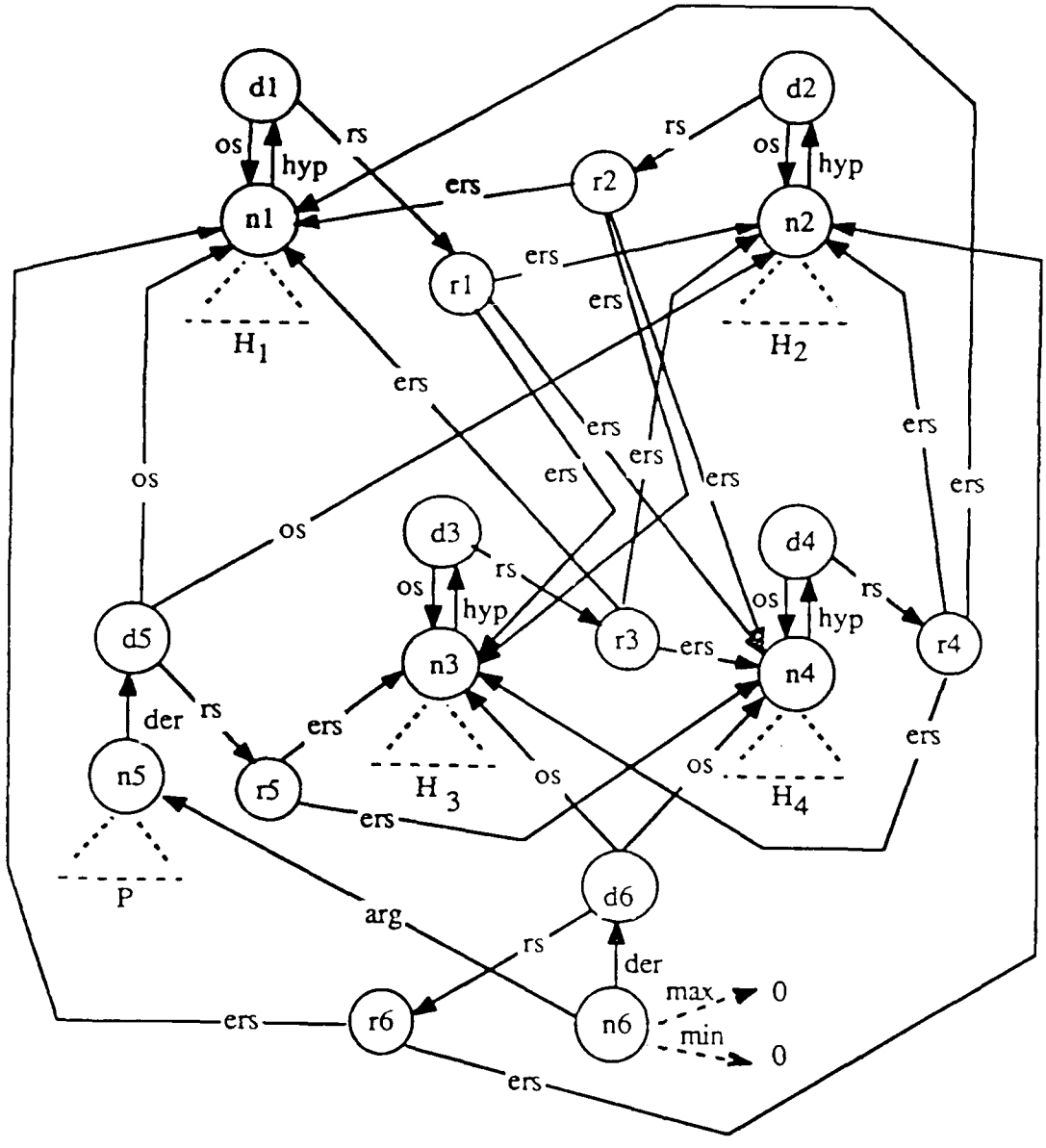


Figure 11
Network after URS

There are two points about the application of URS that are worth discussing. The first one concerns the case in which one of the contradictory nodes was obtained by more than one derivation (i.e., its support contains more than one supporting node). An example of this situa-

tion is represented in Figure 12.

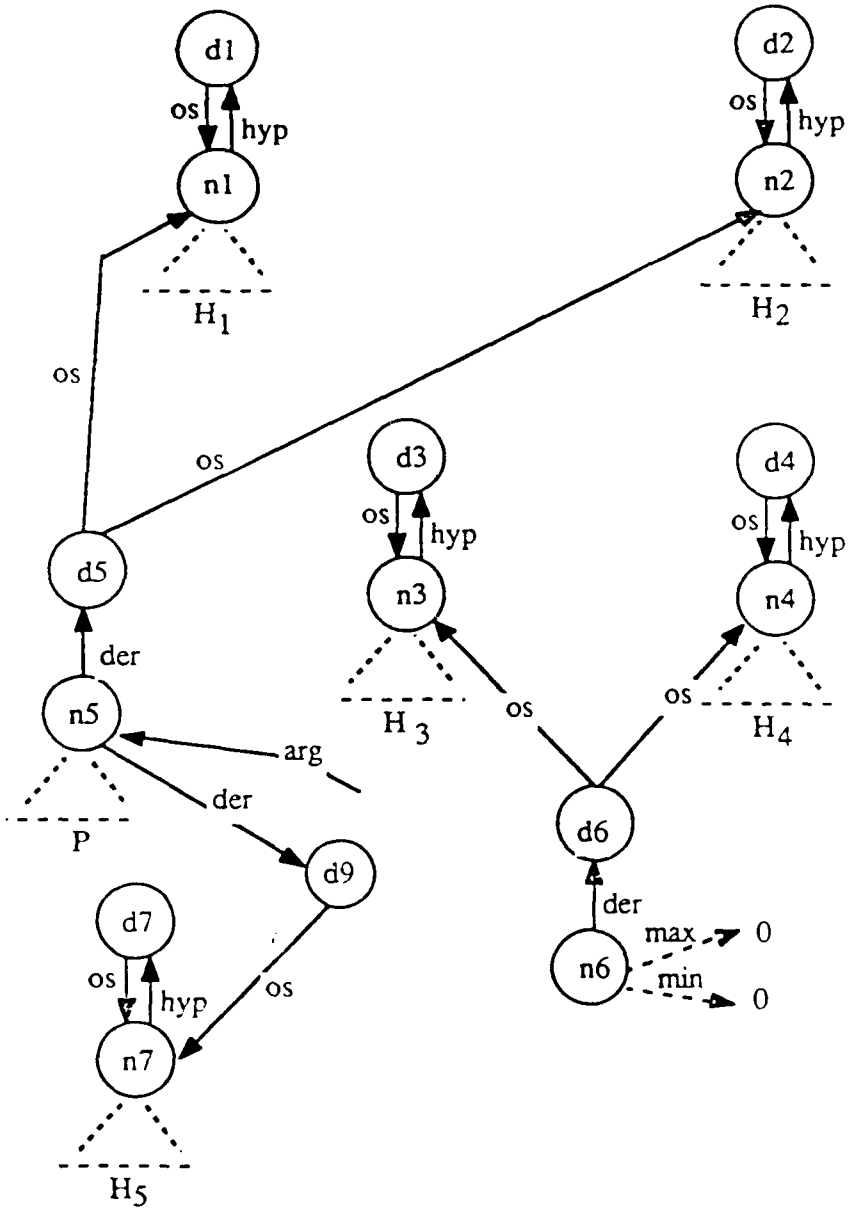


Figure 12
Network before URS (node n5 has two supporting nodes)

In this case, since n5 has two supporting nodes (d5 and d8), when n6 is derived the inference system does not uncover just one inconsistent set but rather two: the sets of hypotheses $\{H_1$,

H_2, H_3, H_4} and {H_3, H_4, H_5} (represented by the nodes {n1, m2, n3, n4} and {n3, n4, n7}) are now known to be inconsistent. The application of URS results in the network represented in Figure 13.

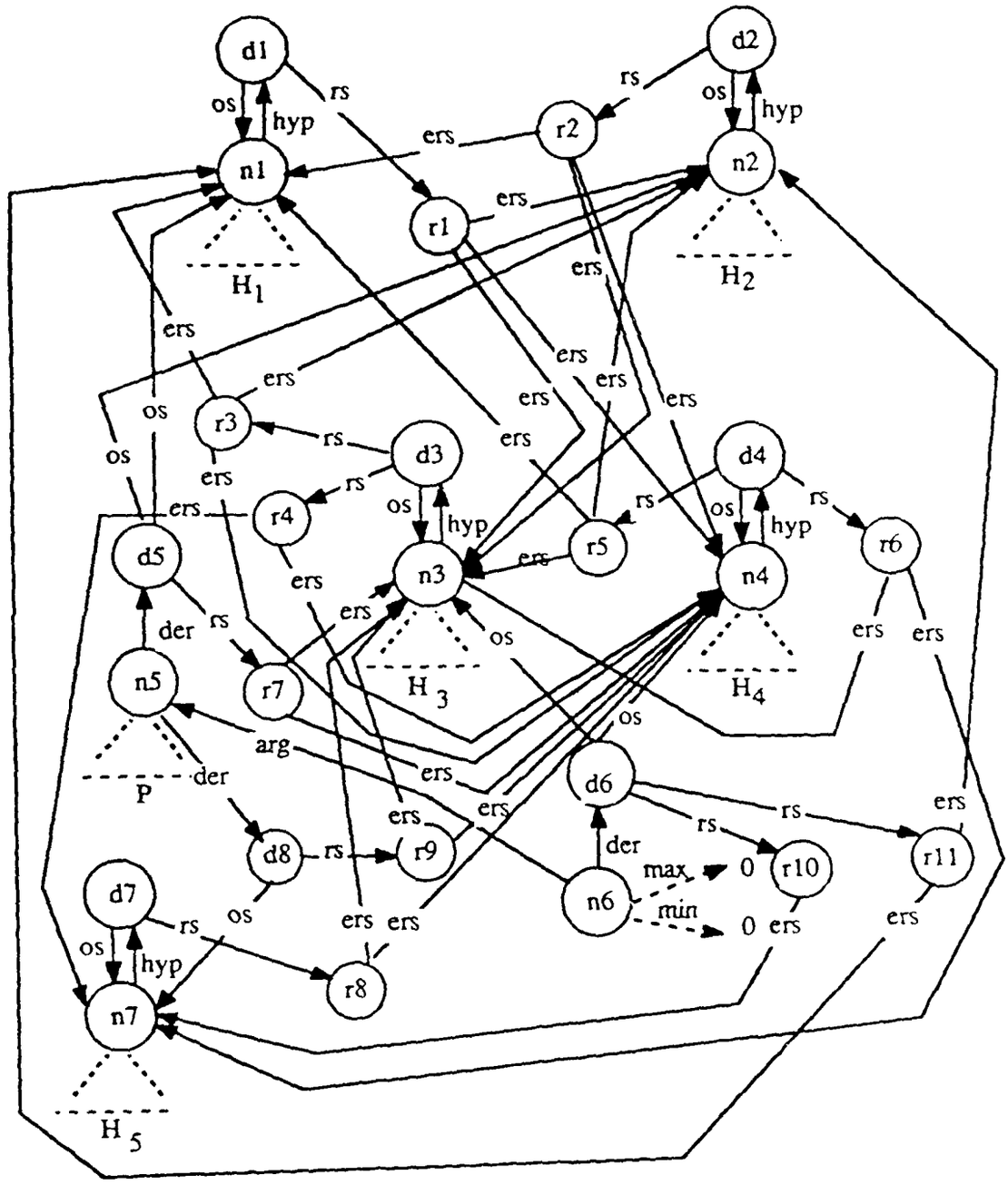


Figure 13
Network after URS (node n5 has two supporting nodes)

The second case worth discussing concerns simplification of RSs. Recall that the rule of URS states that upon discovery of the inconsistent set φ, all the wffs that have an OS that is

not disjoint from φ should have their RSs updated. If $F \mid t,o,r$ is a wff such that $o \cap \varphi \neq \emptyset$, then after application of URS this wff becomes $F \mid t,o,\sigma(r \cup (\varphi - o))$. If r is the empty set, as was the case in the previous examples, then a new set is created in the wff's RS. However, if r is not the empty set, the resulting RS may turn out to be smaller than the RS before the application of URS. An example of such case is presented next. Consider the network of Figure 11 and suppose that an alternative derivation for n5 is discovered, involving only the hypothesis H_1 (node d7, Figure 14). This uncovers the inconsistent set $\{H_1, H_3, H_4\}$, represented in the network by the nodes n1, n3 and n4. In this case, the application of URS has the effect of decreasing the size of some RSs (namely r1, r3 and r4), as represented in Figure 14.

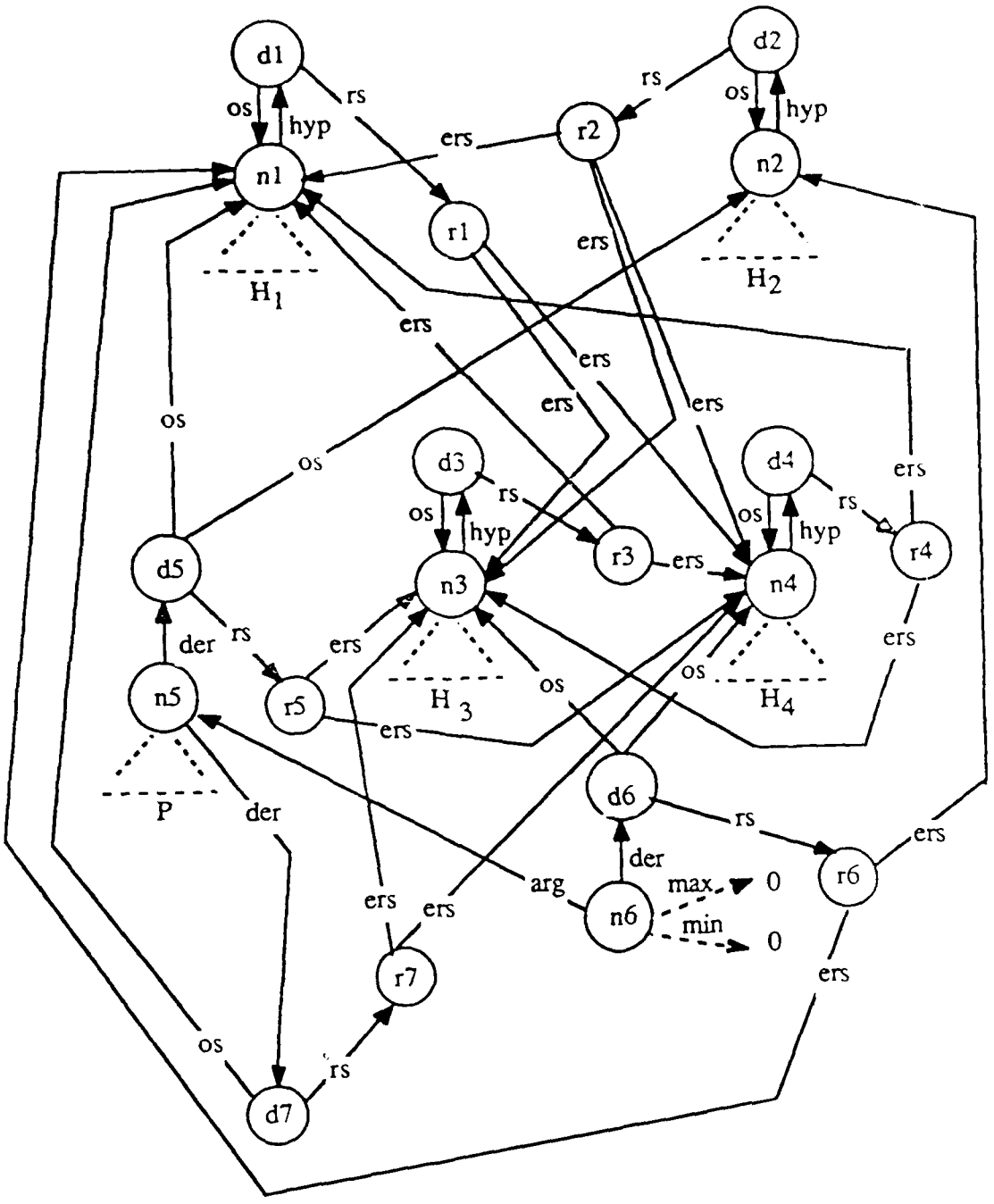


Figure 14
Network after URS (some RSs decreased in size)

8. Selecting Among Alternatives

In this section, we present an example of person-machine interaction by showing a run in which SNeBR obtains a solution to a puzzle, "The Woman Freeman Will Marry", from [Summers 72]. A characteristic of this puzzle is that there is no straightforward path from the pro-

positions in the puzzle's statement to the puzzle's solution. In solving this puzzle, one usually raises hypotheses, reasons from them, and, if a contradiction is detected, identifies the faulty hypotheses, replaces some of them, and resumes the reasoning. SNeBR is adequate for the solution of this type of puzzle, since it allows the introduction of hypotheses, reasoning from them, and, when contradictions are detected, the detection of *exactly* which hypotheses led to the contradiction and the retraction of some of the hypotheses introduced.

The statement of the puzzle is as follows:

> Freeman knows five women: Ada, Bea, Cyd, Deb and Eve. The women are in two age brackets: three women are under 30 and two women are over 30. Two women are teachers and the other three women are secretaries. Ada and Cyd are in the same age bracket. Deb and Eve are in different age brackets. Bea and Eve have the same occupation. Cyd and Deb have different occupations. Of the five women, Freeman will marry the teacher over 30. Who will Freeman marry?

To use SNeBR to solve this puzzle, the propositions referenced in the puzzle statement have to be represented in the network. What follows is a description of every proposition referenced in the puzzle's statement. The numbers associated with the wffs relate to the number of the node that represents the wff in the network. The wffs are described in a language called SNePSLOG [McKay and Martins 81] which is a logic programming interface to SNePS. Assertions and rules written in SNePSLOG are stored as structures in the SNePS network; SNePSLOG queries are translated into top-down deduction requests to SNeBR; output from SNeBR is translated into SNePSLOG formulas.

There are five women, Ada, Bea, Cyd, Deb, and Eve.

wff1 : *Woman*(*Ada*)

wff2 : *Woman*(*Bea*)

wff3 : *Woman*(*Cyd*)

wff4 : *Woman*(*Deb*)

wff5 : *Woman*(*Eve*)

The women are in two age brackets: three women are under 30 (wff12) and two women are over 30 (wff18). It is implicit in the statement of the puzzle that every woman is either under 30 or over 30 (wff27).

wff12 : $_{5}\mathbb{W}_{3}^{3}$ (*age*(*Ada*,*u*30),*age*(*Bea*,*u*30),*age*(*Cyd*,*u*30),*age*(*Deb*,*u*30),*age*(*Eve*,*u*30))[51]

wff18 : $_{5}\mathbb{W}_{2}^{2}$ (*age*(*Ada*,*o*30),*age*(*Bea*,*o*30),*age*(*Cyd*,*o*30),*age*(*Deb*,*o*30),*age*(*Eve*,*o*30))

wff27 : $\forall x$*Woman*(x)$\rightarrow_{2}\mathbb{W}_{1}^{1}$(*age*($x$,*u*30),*age*($x$,*o*30))

Two women are teachers (wff33) and the other three women are secretaries (wff39). The *the* in the previous sentence conveys the information that no woman is both a teacher and a secretary, represented by wff48.

wff33 : $_{5}\mathbb{W}_{2}^{2}$(*worker*(*Ada*,*teacher*),*worker*(*Bea*,*teacher*),*worker*(*Cyd*,*teacher*),
worker(*Deb*,*teacher*),*worker*(*Eve*,*teacher*))

wff39 : $_{5}\mathbb{W}_{3}^{3}$(*worker*(*Eve*,*secretary*),*worker*(*Deb*,*secretary*),*worker*(*Cyd*,*secretary*),
worker(*Bea*,*secretary*),*worker*(*Ada*,*secretary*))

wff48 : $\forall(x)$*Woman*(x)$\rightarrow_{2}\mathbb{W}_{1}^{1}$(*worker*($x$,*secretary*),*worker*($x$,*teacher*))

Ada and Cyd are in the same age bracket (wff53).

wff53 : $\forall(x)_{2}\mathcal{O}_{1}$(*age*(*Ada*,$x$),*age*(*Cyd*,$x$))

Deb and Eve are in different age brackets.

wff58 : $\forall(x)_{2}\mathbb{W}_{1}^{1}$(*age*(*Deb*,$x$),*age*(*Eve*,$x$))

---

[51]With this proposition we can see the advantage of the SNePS connectives. With the standard connectives this proposition would have to be expressed in the following way: (-age(Ada,u-30) ∧ -age(Bea,u-30) ∧ age(Cyd,u-30) ∧ age(Deb,u-30) ∧ age(Eve,u-30))∨ (-age(Ada,u-30) ∧ age(Bea,u-30) ∧ -age(Cyd,u-30) ∧ age(Deb,u-30) ∧ age(Eve,u-30))∨ -(age(Ada,u-30) ∧ age(Bea,u-30) ∧ age(Cyd,u-30) ∧ -age(Deb,u-30) ∧ age(Eve,u-30))∨ (-age(Ada,u-30) ∧ age(Bea,u-30) ∧ age(Cyd,u-30) ∧ age(Deb,u-30) ∧ -age(Eve,u-30))∨ (age(Ada,u-30) ∧ -age(Bea,u-30) ∧ -age(Cyd,u-30) ∧ age(Deb,u-30) ∧ age(Eve,u-30))∨ (age(Ada,u-30) ∧ -age(Bea,u-30) ∧ age(Cyd,u-30) ∧ -age(Deb,u-30) ∧ age(Eve,u-30))∨ (age(Ada,u-30) ∧ age(Bea,u-30) ∧ -age(Cyd,u-30) ∧ -age(Deb,u-30) ∧ age(Eve,u-30))∨ (age(Ada,u-30) ∧ age(Bea,u-30) ∧ -age(Cyd,u-30) ∧ age(Deb,u-30) ∧ -age(Eve,u-30))∨ (age(Ada,u-30) ∧ age(Bea,u-30) ∧ age(Cyd,u-30) ∧ -age(Deb,u-30) ∧ -age(Eve,u-30))

Bea and Eve have the same occupation.

$$wff63 : \forall(x)_2\Theta_1(worker(Bea,x),worker(Eve,x))$$

Cyd and Deb have different occupations.

$$wff68 : \forall(x)_2\mathbb{X}_1^1(worker(Cyd,x),worker(Deb,x))$$

Exactly one woman over 30 is a teacher.

$$\begin{aligned} wff79 : \ & {}_5\mathbb{X}_1^1(\ {}_2\mathbb{X}_2^2(age(Ada,o30),worker(Ada,teacher)), \\ & {}_2\mathbb{X}_2^2(age(Bea,o30),worker(Bea,teacher)), \\ & {}_2\mathbb{X}_2^2(age(Cyd,o30),worker(Cyd,teacher)), \\ & {}_2\mathbb{X}_2^2(age(Deb,o30),worker(Deb,teacher)), \\ & {}_2\mathbb{X}_2^2(age(Eve,o30),worker(Eve,teacher))) \end{aligned}$$

Freeman will marry the teacher over 30.

$$wff88 : \forall(x)_2\Theta_1(marry(Freeman,x),{}_2\mathbb{X}_2^2(age(x,o30),worker(x,teacher)))$$

Using the propositions described above, we build into the network the hypotheses represented in Figure 15.

$$wff6 : \ {}_5\mathbb{X}_5^5(wff5,wff4,wff3,wff2,wff1) \mid hyp,\{wff6\},\{\}$$

$$\begin{aligned} wff89 : \ & {}_{12}\mathbb{X}_{12}^{12}(wff88,wff79,wff68,wff63,wff58,wff53,wff48,wff39,wff33, \\ & wff27,wff18,wff12) \mid hyp,\{wff89\},\{\} \end{aligned}$$

Figure 15
Hypotheses representing the puzzle's statement

The hypothesis represented by wff6 states that there are five women and names those women, and the hypothesis represented by wff89 asserts all the specific information pertaining to these

women and their relationship with Freeman.

To solve the puzzle, we raise hypotheses about the ages and professions of the women and ask SNeBR to deduce whom Freeman will marry under those assumptions.[52] If the hypotheses raised are consistent with the puzzle's statement the desired answer will be returned; otherwise, a contradiction will be detected, and SNeBR will guide us in selecting new hypotheses.

wff13 : *age*(*Ada*,*o30*) | hyp,{wff13} ,{}

wff15 : *age*(*Cyd*,*o30*) | hyp,{wff15} ,{}

wff28 : *worker*(*Ada*,*teacher*) | hyp,{wff28} ,{}

wff31 : *worker*(*Deb*,*teacher*) | hyp,{wff31} ,{}

Figure 16
Hypotheses raised

Suppose that we raise the hypotheses represented in Figure 16. These hypotheses specify which women are supposed to be over 30 and which women are supposed to be teachers. All these hypotheses plus the hypotheses shown in Figure 15 were built into the network.

Suppose that we ask who Freeman will marry in the BS defined by the context {wff6, wff13, wff15, wff28, wff31 and wff89}. We present next some of the results generated following this query.

[52]It should be noticed that specifying the ages of the two women over 30 completely determines the ages of the five women and that specifying the names of the two women who are teachers completely determines the profession of the five women.

I wonder if $marry(Freeman,who)$
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

let me try to use the rule $\forall(x)_2 \Theta_1(marry(Freeman,x),_2 \aleph_2^2 (age(x,o30),worker(x,teacher)))$

Figure 17
Who will Freeman marry?

In the BS under consideration, there is no assertion about who Freeman will marry, but wff88 may enable its deduction. SNeBR sets up two sub-goals, finding who is over 30 and finding who is a teacher (Figure 18).

I wonder if $age(x,o30)$
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

I know $age(Cyd,o30)$

I know $age(Ada,o30)$

I wonder if $worker(x,teacher)$
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

I know $worker(Deb,teacher)$

I know $worker(Ada,teacher)$

Figure 18
Ada and Cyd are over 30
Ada and Deb are teachers

since $worker(Ada,teacher)$ and $age(Ada,o30)$

I infer $marry(Freeman,Ada)$

Figure 19
Freeman will marry Ada

Figure 19, shows SNeBR's deduction that Freeman will marry Ada. The inference does not stop here, however, since there are several processes still waiting for answers and SNeBR

reports inferences as shown in Figure 20.

since $age(Ada,o30)$ and $age(Cyd,o30)$

I infer $_1\mathbb{X}_0^0(age(Eve,o30))$

since **not** $age(Eve,o30)$

I infer $age(Deb,o30)$

Figure 20
Deb is over 30

Taking into account the information gathered in Figure 20 a contradiction is detected (Figure 21). In fact, Figure 20 shows the inference that Deb is over 30. This information is supplied to the process working on the rule $_5\mathbb{X}_3^3(age(Ada,o30), age(Bea,o30), age(Cyd,o30), age(Deb,o30), age(Eve,o30))$ which gathers that there are three women who are over 30 (Ada, Cyd and Deb), which contradicts the rule.

WARNING!
Contradiction detected in the following and-or
$_5\mathbb{X}_3^3(age(Ada,o30),age(Bea,o30),age(Cyd,o30),age(Deb,o30),age(Eve,o30))$
More true arguments than max.
Arguments in wrong number $age(Ada,o30)$ $age(Cyd,o30)$ $age(Deb,o30)$
You have the following options:
1. Continue anyway, knowing that a contradiction is derivable;
2. Re-start the exact same request in a different context which is not inconsistent;
3. Drop the request altogether.
Do you want to continue anyway?
=><= n
Do you want to re-start the request in a new context?
=><= yes

Figure 21
A contradiction is detected

Upon detecting the contradiction SNeBR gives the options of continuing the reasoning within the inconsistent belief space, of modifying the current context in order to obtain a consistent context (and therefore a consistent BS) or of giving up the request. We decide to restore

consistency causing the interaction shown in Figures 22 and 23.

```
In order to make the context consistent you must delete one hypothesis from the set
(wff13 wff15 wff89)

Do you want to take a look at wff13 ?
=><= n
There are 5 propositions depending on wff13 : (wff97 wff16 wff93 wff91 wff90) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= a

  1 ⊗0 (marry(Freeman,Eve)) | ext,{wff13,wff28,wff89} ,{{wff15}}
                .
                .

What do you want to do with wff13 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= d

Do you want to take a look at wff15 ?
=><= y
  age(Cyd,o30) | hyp,{wff15} ,{{wff13,wff89}}
There are 2 propositions depending on wff15 : (wff16 wff91) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= n
What do you want to do with wff15 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= d

Do you want to take a look at wff89 ?
=><= n
There are 8 propositions depending on wff89 :
(wff97 wff95 wff16 wff94 wff93 wff92 wff91 wff90) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= n
What do you want to do with wff89 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= k
```

Figure 22
Inspecting the inconsistent set of hypotheses

Figure 22 shows the inspection of the hypotheses that are responsible for the contradiction. Notice that although the current context is the set {wff6, wff13, wff15, wff28, wff31, wff89}, only the hypotheses represented by wff13, wff15, and wff89 were used in the derivation of the contradiction, and thus they are the only ones that *must* be changed in order to

restore consistency, i.e., the only ones whose change may restore consistency. It should be further noticed that SWM guarantees that removing *exactly* one of them will generate a set that is not known to be inconsistent. We keep the hypothesis concerning the statement of the puzzle (wff89) and discard the hypotheses concerning the women's ages (wff13 and wff15). We also enter new hypotheses concerning the women's ages (Figure 23).

```
The following (not known to be inconsistent) set of hypotheses was also part of the
context where the contradiction was derived: (wff31 wff28 wff6)
Do you want to inspect or discard some of them?
=><= n

Do you want to add some new hypotheses?
=><= y
Enter an hypothesis using SNePSLOG
=><= age(Bea,o30)
Do you want to enter another hypothesis?
=><= y
Enter an hypothesis using SNePSLOG
=><= age(Deb,o30)
Do you want to enter another hypothesis?
=><= n
```

Figure 23
Adding new hypotheses

After resolving the contradiction, the inferences resumes (Figures 24 and 25). In this case there are no further contradictions detected and SNeBR reports that Freeman will marry Deb and will not marry Ada, Bea, Cyd or Eve.

I wonder if *marry*(*Freeman*,*who*)
holds within the BS defined by the context (wff14 wff16 wff6 wff28 wff31 wff89)

I know *age*(*Deb*,*o30*)

I know *age*(*Bea*,*o30*)

I know *worker*(*Deb*,*teacher*)

I know *worker*(*Ada*,*teacher*)

Figure 24
Resuming the deduction

since $_2\mathbb{X}_2^2$(*age*(*Deb*,*o30*),*worker*(*Deb*,*teacher*))

I infer *marry*(*Freeman*,*Deb*)

Figure 25
Freeman will marry Deb

Some of the propositions returned are shown in Figure 26. It should be noticed that there are two ways of deducing the second wff of Figure 26 and thus it has two supports, one for each of the possible derivations.

marry(*Freeman*,*Deb*) | ext,{wff16,wff31,wff89} ,{{wff13,wff15}}

$_1\mathbb{X}_0^0$(*marry*(*Freeman*,*Eve*))
| ext,{wff16,wff31,wff89} ,{{wff13,wff15}}
| ext,{wff13,wff28,wff89} ,{{wff15}}

Figure 26
Some of the nodes returned

9. Concluding Remarks

In this paper, we discussed an important class of AI programs, belief revision systems. Belief revision is important whenever reasoning is performed with a knowledge base that may contain contradictory information. Belief revision systems are capable of considering only part of the knowledge base (the set of ***believed propositions***), perform inferences from this set, and, if a contradiction is detected, replace this set by another one (*change their beliefs*), disregarding every proposition that does not belong to the new set. To obtain this behavior, belief revision systems have to maintain a record of where each proposition in the knowledge base came from. We discussed the two ways of keeping these records, corresponding to assumption-based and justification-based systems. We evaluated both approaches, concluding that the assumption-based approach presents several advantages over the justification-based one.

In order to build a robust assumption-based belief revision system, we developed a formalism that associates each proposition in the knowledge base with the smallest set of hypotheses used in its derivation. We presented a logic (SWM) loosely based on relevance logic that captures the notion of propositional dependency and is able to deal with contradictions. SWM associates two sets with each proposition: the *origin set* contains *every* hypothesis used in the derivation of the proposition; the *restriction set* contains those sets of hypotheses that are incompatible with the proposition's origin set.

SWM's rules of URS and $\neg$I handle contradictions. Upon detection of a contradiction and identification of the hypotheses contained in the origin sets of the contradictory propositions as an inconsistent set, the rule of URS has the effect of recording the inconsistent set in the restriction set of every proposition depending on hypotheses from this set. The rule of $\neg$I allows the derivation of new propositions from the occurrence of a contradiction.

Each proposition generated by the rules of inference of SWM has a *minimal* restriction set, in the sense that restriction sets are free from some kinds of redundancies. Each such proposition has a *maximal* restriction set in the sense that its restriction set records all inconsistent

sets *known* so far. Every proposition with the same origin set has the same restriction set, reflecting the fact that restriction sets are both minimal and maximal.

We defined the behavior of an abstract program based on SWM, the Multiple Belief Reasoner (MBR). In MBR, a *context* is any set of hypotheses. A context determines a *belief space* (BS), which is the set of all propositions whose origin set is contained in the context. In other words, a BS contains all the propositions that depend exclusively on the hypotheses defining the context. Given any context, the only propositions whose truth value is *known* are those propositions that belong to the BS defined by the context. The truth value of all the other propositions is unknown. By a proposition having an unknown truth value, we mean that in order to compute its truth value one has to carry out further deduction, and it may even be possible that its truth value is not computable from the hypotheses under consideration. At any moment, the only propositions that are believed (and thus retrievable from the knowledge base) are the ones that belong to the BS under consideration.

We discussed the applicability, in MBR, of the rules of inference of SWM that deal with contradictions: when a contradiction is found and only one of the contradictory propositions belongs to the BS under consideration, then URS must be applied, recording the contradiction, and $\neg$I is not applicable. If both contradictory propositions belong to the BS under consideration, then URS must be applied, and $\neg$I should be applied if one decides to get rid of the contradiction.

When a contradiction is detected in the BS under consideration, and, after selecting some hypotheses (or just one hypothesis) as the culprit for the contradiction, when one is faced with the problem of making inaccessible to the belief revision system all the propositions that were previously derived from such hypotheses, all one has to do in MBR is remove the selected hypotheses from the context under consideration. Afterwards, all the propositions derived from the selected hypotheses are no longer in the BS under consideration and consequently are not retrievable by the deduction system.

We discussed, SNeBR, a particular implementation of MBR using SNePS. Finally, we presented the output produced by SNeBR when trying to find the solution of a puzzle. Our example shows that, after detection of a contradiction, SNeBR allows the identification of ***exactly*** which hypotheses contributed to the contradiction and allows the inspection of the consequences of removing each one of those hypotheses. Furthermore, the SWM formalism that underlies SNeBR, guarantees that the removal of exactly one of those hypotheses (no matter which) produces a context that is not known to be inconsistent.

10. Acknowledgments

Many thanks to John Corcoran, Jon Doyle, Donald McKay, Ernesto Morgado, J. Terry Nutter, William J. Rapaport and members of the SNePS Research Group for their criticisms and suggestions.

This work was partially supported by the National Science Foundation under Grant MCS80-06314 and by the Instituto Nacional de Investigacão Cientifica (Portugal), under Grant No. 20536; Preparation of this paper was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008.

11. References

Anderson A. and Belnap N., **Entailment: The Logic of Relevance and Necessity** Vol.1, Princeton University Press, 1975.

Charniak E., Riesbeck C. and McDermott D., **Artificial Intelligence Programming**, Lawrence Erlbaum Associates, 1980.

de Kleer J., "Choices without Backtracking", **Proc. AAAI-84**, pp. 79-85.

Doyle J., "Truth Maintenance Systems for Problem Solving", Technical Report AI-TR-419, Massachussets Institute of Technology, AI Lab., 1978

Doyle J., "A Truth Maintenance System", **Artificial Intelligence**, Vol.12 No.3, pp.231-272, 1979.

Doyle J., "A model for Deliberation, Action and Introspection", Ph.D. Dissertation, Technical Report No.581, Massachussets Institute of Technology, AI Lab, 1980.

Doyle J., "Some Theories of Reasoned Assumptions: An Essay in Rational Psychology", Department of Computer Science, Carnegie-Mellon University, 1982.

Doyle J., "The Ins and Outs of Reason Maintenance", **Proc. IJCAI-83**, pp.349-351.

Doyle J. and London P., "A selected descriptor-indexed bibliography to the literature of Belief Revision", **SIGART Newsletter 71**, pp.7-23, 1980.

Fahlman S., "A Planning System for Robot Construction Tasks", **Artificial Intelligence**, Vol.5, no.1, pp.1-49, 1974.

Fikes R., "Deductive Retrieval Mechanisms for State Description Models", **Proc. IJCAI-75**, pp.99-106.

Fikes R. and Nilson N., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", **Artificial Intelligence**, Vol.2, No.3-4, pp.189-208, 1971.

Fitch F., **Symbolic Logic: An Introduction**, Ronald Press, 1952.

Goodwin J. W., "A Improved Algorithm for Non-Monotonic Dependency Net Update", Research Report LiTH-MAT-R-82-23, Software Systems Research Center, Linkoping Institute of Technology, Sweden, August 1982.

Goodwin J.W., "WATSON: A Dependency Directed Inference System", **Proc. Non-Monotonic Reasoning Workshop**, pp.103-114, 1984.

Haack S., **Philosophy of Logics**, Cambridge University Press, 1978.

Hayes P.J., "The Frame Problem and Related Problems in Artificial Intelligence", in **Artificial and Human Thinking**, Elithorn and Jones (eds.), pp.45-59, Jossey-Bass Inc., 1973.

Hewitt C., "Description and Theoretical Analysis of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot", Technical Report TR-258, Massachussets Institute of Technology, 1972.

Kaplan R., "A Multi-processing Approach to Natural Language", **Proc. National Computer Conference**, pp.435-440, 1973.

Lemmon E.J., **Beginning Logic**, Hackett Publishing Company, 178.

London P., "Dependency Networks as Representation for Modelling in General Problem Solvers", Ph.D. Dissertation, Technical Report 698, Department of Computer Science, University of Maryland, 1978.

Maida A. and Shapiro S., "Intensional Concepts in Propositional Semantic Networks", **Cognitive Science**, Vol.6, No.4, pp.291-330, 1982.

Martins J., "Reasoning in Multiple Belief Spaces", Ph.D. Dissertation, Technical Report 203, Department of Computer Science, State University of New York at Buffalo, 1983.

Martins J., "Belief Revision", in **Encyclopedia of Artificial Intelligence**, Shapiro (ed.), John Wiley and Sons, forthcoming.

Martins J. and Shapiro S., "Reasoning in Multiple Belief Spaces", **Proc. IJCAI-83**, pp.370-373.

Martins J. and Shapiro S., "A Model for Belief Revision", **Proc. Non-Monotonic Reasoning Workshop**, pp.241-294, American Association for Artificial Intelligence, 1984.

Martins J. and Shapiro S., "A Logic for Belief Revision", forthcoming.

McAllester D., "A Three-valued Truth Maintenance System", Technical Report Memo 473, Massacussets Institute of Technology, AI Lab., 1978.

McAllester D., "An Outlook on Truth-Maintenance", AI Memo 551, Massachussets Institute of Technology, AI Lab., 1980.

McCarthy J. and Hayes P., "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in **Machine Intelligence 4**, Meltzer and Michie (eds.), pp.463-502, Edinburgh University Press, 1969.

McDermott D., "Contexts and Data Dependencies: A Synthesis", Department of Computer Science, Yale University, 1982.

McDermott D., "Data Dependencies on Inequalities", **Proc. AAAI-83**, pp.266-269.

McDermott D. and Sussman G., "The CONNIVER Reference Manual", Technical Report Memo 259, Massachussets Institute of Technology, 1972.

McKay D and Martins J., "Provisional SNePSLOG User's Manual", SNeRG Technical Note 4, Department of Computer Science, State University of New York at Buffalo, 1981.

McKay D. and Shapiro S., "MULTI – A LISP Based Multiprocessing System", **Proc. 1980 LISP Conference**, pp.29-37.

Raphael B., "The Frame Problem in Problem Solving Systems", in **Artificial Intelligence and Heuristic Programming**, Findler and Meltzer (eds.), pp.159-169, American Elsevier, 1971.

Rulifson J., Derksen J. and Waldinger R., "QA4: A Procedural Calculus for Intuitive Reasoning", Technical Report Note 73, SRI International, 1972.

Saks V., "A Matcher for Intensional Semantic Networks", unpublished manuscript, Department of Computer Science, State University of New York at Buffalo, 1985.

Shapiro S., "Representing and Locating Deduction Rules in a Semantic Network", in Proc. Workshop on Pattern-Directed Inference Systems, **SIGART Newsletter 63**, pp.14-18, 1977.

Shapiro S., "The SNePS Semantic Network Processing System", in **Associative Networks**, Findler (ed.), pp.179-203, Academic Press, 1979a.

Shapiro S., "Using Non-standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network", presented at "Current Aspects of AI Research", a seminar held at the Electrotechnical Laboratory, Tokyo, August 27-28, 1979b.

Shapiro S., Martins J. and McKay D., "Bi-Directional Inference", **Proc. of the Fourth Annual Conference of the Cognitive Science Society**, pp.90-93, 1982.

Shapiro S. and Wand M., "The Relevance of Relevance", Technical Report No.46, Computer Science Department, Indiana University, 1976.

Shrobe E., "Dependency-directed Reasoning in the Analysis of Programs which Modify Complex Data Structures", **Proc. IJCAI-79**, pp.829-835.

Stallman R. and Sussman G., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", **Artificial Intelligence**, Vol.9, No.2, pp.135-196, 1977.

Sussman G., Winograd T. and Charniak E., "MICRO-PLANNER Reference Manual", Technical Report Memo 203, Massachussets Institute of Technology, 1971.

Summers G., **Test your Logic**, Dover Publications, 1972.

Thompson A., "Network Truth-Maintenance for Deduction and Modelling", **Proc. IJCAI-79**, pp.877-879.

Wand M., "The Frame Model of Computation", Technical Report No.20, Computer Science Department, Indiana University, Indiana, 1974.

Appendix 1: Theorems about the SWM System

In this appendix we present some of the properties of the SWM system.

Theorem 1: Given the supported wffs $A \mid t_a,o_a,r_a$ and $B \mid t_b,o_b,r_b$ then the supported wff $C \mid t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ has a minimal RS, regardless of whether or not A and B have minimal RSs.

Proof: Suppose by way of contradiction that given $A \mid t_a,o_a,r_a$ and $B \mid t_b,o_b,r_b$ the supported wff $C \mid t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ does not have a minimal RS. This entails that one of the following two cases holds:

1. $\exists r \in \mu(\{r_a,r_b\},\{o_a,o_b\}) : r \cap (o_a \cup o_b) \neq \emptyset$
2. $\exists r,s \in \mu(\{r_a,r_b\},\{o_a,o_b\}) : r \subset s$.

Suppose that condition 1 holds. Since $r \in \mu(\{r_a,r_b\},\{o_a,o_b\})$ we can infer that $r \in \Psi(r_a \cup r_b,o_a \cup o_b)$, which, due to the way Ψ was defined, contradicts the fact that $r \cap (o_a \cup o_b) \neq \emptyset$.

Suppose that condition 2 holds. Since $r,s \in \mu(\{r_a,r_b\},\{o_a,o_b\})$ we can infer that $r,s \in \sigma(\Psi(r_a \cup r_b,o_a \cup o_b))$, which, due to the way σ was defined, contradicts the fact that $r \subset s$.

Therefore $C \mid t_c,o_a \cup o_b,\mu(\{r_a,r_b\},\{o_a,o_b\})$ has minimal RS. □

Theorem 2: Given the supported wffs $A \mid t_a,o_a,\{R_1, \ldots, R_n\}$ and $B \mid t_b,o_b,r_b$ then $C \mid t_c,o_a - o_b,(o_a - o_b)$ has minimal RS.

Proof: Suppose that $o_a - o_b = \{H_1, \ldots, H_n\}$. Taking into account that $C \mid t_c,o_a - o_b,(o_a - o_b)$ is the same as $C \mid t_c,o_a - o_b,\mu(\mathbf{rs}(H_1), \ldots, \mathbf{rs}(H_n)),(\{H_1\}, \ldots, \{H_n\}))$ the statement of this theorem follows from Theorem 1. □

Theorem 3: If A is an inconsistent set then so is any set containing A.

Proof: The proof presented is based on the fact that a proof of B from $\{A_1, \ldots, A_n\}$ is a sequence of lines, the first n of which are $A_1, \ldots, A_n$ and the last of which is B. Each line between A_n and B is obtained from the previous line(s) by the application of some rule of inference and is justified by $R(L_1,L_2)$ in which R represents some rule of inference and (L_1,L_2) are the line numbers of the parent wffs.

Suppose that $A=\{P_1, \ldots, P_n\}$ and let $r_1(L_{11},L_{12})$, $r_2(L_{21},L_{22})$, $\ldots$, $r_k(L_{k1},L_{k2})$ represent the sequence of applications of rules of inference to the (ordered) set A (and to the wffs derived from them) to generate a $\rightarrow\leftarrow$. Suppose that $A \subset B$ and $B-A=\{S_1, \ldots, S_i\}$. B can, therefore, be written as the following ordered set (in which the order of $S_1, \ldots, S_i$ is irrelevant) $B=\{S_1, \ldots, S_i, P_1, \ldots, P_n\}$. Then, letting $L'_{1g}=L_{1g}$, the following sequence of rules of inference $r_1(L'_{11},L'_{12})$, $r_2(L'_{21},L'_{22})$, $\ldots$, $r_k(L'_{k1},L'_{k2})$ describes a derivation of a $\rightarrow\leftarrow$ from B. $\square$

Corollary 3.1: If A is not known to be inconsistent then neither is any set contained in A.

Proof: Suppose by way of contradiction that A is not known to be inconsistent and that $B \subset A$ is known to be inconsistent. By the Theorem 3, A is known to be inconsistent, which is a contradiction. $\square$

Theorem 4: All the supported wffs in the knowledge base resulting from the application of the rules of inference of the SWM system have minimal RSs.

Proof: The proof will be done by complete induction on the number of applications of rules of inference.

The only supported wffs which can be obtained by applying one rule of inference only are hypotheses and the rule of Hyp guarantees that they have minimal RSs.

Suppose now, by induction hypothesis, that all the supported wffs obtained by the application of less than *n* rules of inference have minimal RSs. We will have to prove that the supported wff obtained by the application of *n* rules of inference has

minimal RSs as well.

1. The supported wff is obtained by the application of either $\neg$E, $\wedge$E, $\vee$I, $\exists$I, or $\exists$E, then it has the same OS and RS as the parent supported wff and consequently has minimal RS.

2. The supported wff is obtained by the application of either MP, MT, $\wedge$I, $\vee$E, or $\forall$E, then by Theorem 1 we may conclude that it has minimal RS.

3. The supported wff is obtained by the application of $\neg$I, or $\forall$I then by Theorem 2 we can conclude that it has minimal RS.

4. The supported wff is obtained by $\neg$I. We will show that a supported wff obtained using the second part of the rule of $\neg$I has minimal RS (to show that a supported wff obtained using the first part of the rule of $\neg$I has minimal RS as well it suffices to take $o_1=o_2=o$ in the proof that follows). Consider $A \mid t_1,o_1,r_1$ and $\neg A \mid t_2,o_2,r_2$. For $\{H_1, \ldots, H_n\} \subset (o_1 \cup o_2)$, the second part of the rule of $\neg$I allows us to deduce $\neg(H_1 \wedge \ldots \wedge H_n) \mid ext, o_1 \cup o_2, (o_a \cup o_b - \{H_1, \ldots, H_n\})$. To show that this supported wff has minimal RS let us suppose that it is obtained in two steps: first, deduce $A \wedge \neg A \mid ext, o_1 \cup o_2, \mu(\{r_1,r_2\},\{o_1,o_2\})$; second, from the above supported wff deduce $\neg(H_1 \wedge \ldots \wedge H_n) \mid ext, o_1 \cup o_2 - \{H_1, \ldots, H_n\}, (o_a \cup o_b - \{H_1, \ldots, H_n\})$, by Theorem 2 this supported wff has minimal RS.

5. The supported wff is obtained by URS. We will consider two sub-cases:

   a. The supported wff results from updating the RS of an hypothesis. Suppose that the supported wff is obtained by updating $H \mid hyp,\{H\},R$. We know that prior to updating H has minimal RS. We want to show that $H \mid hyp,\{H\},\sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ verifies the following conditions:

   1. $\forall r \in \sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ $r \cap \{H\} = \emptyset$

   2. $\forall r,s \in \sigma(R \cup \{(o_1 \cup o_2) - \{H\}\})$ $r \subset s$.

   Condition 1 holds since $\forall r \in R$, $r \cap \{H\} = \emptyset$ (before the update H had minimal RS)

and $[(o_1 \cup o_2)-\{H\}] \cap \{H\} = \emptyset$ (by definition). Condition 2 is verified by definition of σ.

b. The supported wff results from updating the RS of a supported wff whose OT is either "der" or "ext". Suppose that the supported wff is obtained by updating F | t,o,R (t≠hyp). We want to show that F | t,o,σ(R$\cup${$(o_1 \cup o_2)-o$}) verifies the following two conditions:

1. $\forall r \in \sigma(R \cup \{(o_1 \cup o_2)-o\})$ $r \cap o = \emptyset$

2. $\forall r,s \in \sigma(R \cup \{(o_1 \cup o_2)-o\})$ $r \not\subset s$.

Suppose by way of contradiction that condition 1 does not hold. Since F | t,o,R has minimal RS we know that $\forall r \in R$, $r \cap o = \emptyset$, thereby $[(o_1 \cup o_2)-o] \cap o \neq \emptyset$, which is a contradiction, therefore condition 1 holds. Condition 2 follows from the definition of σ.

Therefore all the supported wffs in the knowledge base resulting from the application of the rules of inference of the SWM system have minimal RS. □

Lemma 1: Given n supported wffs $F_1 \mid t_1,o_1,r_1, \ldots, F_n \mid t_n,o_n,r_n$ *then the sets* $R_1=\mu(\{r_1, \ldots ,r_n\},\{o_1, \ldots ,o_n\})$ and $R_2=\mu(\mu(\{r_1, \ldots, r_i\},\{o_1, \ldots ,o_i\}), \mu(\{r_{i+1}, \ldots ,r_n\},\{o_{i+1}, \ldots, o_n\}))$, $\{o_1, \ldots ,o_n\})$ $(1 \leq i \leq n-1)$ *are equal.*

Proof: Suppose by way of contradiction that $R_1 \neq R_2$. This means that either $\exists r \in R_1$ such that $r \notin R_2$ or that $\exists r \in R_2$ such that $r \notin R_1$. We will consider each of these cases in turn:

1. Suppose that $r \in R_1$ and that $r \notin R_2$. Suppose furthermore that r was originated from a set s, i.e., $r=s-(o_1 \cup \ldots \cup o_n)$ and suppose that $s \in r_j$. Letting $O=o_1 \cup \ldots \cup o_n$ we have that r=s−O. The fact that r belongs to R_1 means that $\forall r_k \in \{r_1, \ldots ,r_n\}$ and $r_k \neq r_j$ $\neg(\exists\ u \in r_k : u-O \subset s-O)$. Since $r \notin R_2$, the set s was deleted either by one of the following applications of the operation μ: $\mu(\{r_1, \ldots ,r_i\},\{o_1, \ldots ,r_i\})=R'$ or $\mu(\{r_{i+1}, \ldots$

$\ldots,r_n\},\{o_{j+1},\ldots,o_n\})=R''$, or else s was deleted by $\mu(\{R',R''\},\{o_1,\ldots,o_n\})$.

a. Suppose that s was deleted while creating the set R' (if s was deleted while creating the set R'' the reasoning would be similar) this means that $r_j \in \{r_1,\ldots,r_i\}$. It also means that $\exists r_q \in \{r_1,\ldots,r_i\}$ such that $\exists p \in r_q : p-(o_1 \cup \ldots \cup o_i) \subset s-(o_1 \cup \ldots \cup o_i)$. Therefore $p-O \subset s-O$ which is a contradiction.

b. Suppose that s was deleted while creating R_2, i.e., either $s-(o_1 \cup \ldots \cup o_i) \in R'$ or $s-(o_1 \cup \ldots \cup o_n) \in R''$. In either case, it means that $\exists r_p \in \{r_1,\ldots,r_n\}$ $r_p \neq r_j$ and $\exists u \in r_p : (u-O \subset s-O)$ which is a contradiction.

2. Suppose that $r \in R_2$ and that $r \notin R_1$. Suppose furthermore that r was originated by a set s, i.e., $r=s-O$. Since $r \notin R_1$ this means that $\exists r_k \in \{r_{j+1},\ldots,r_n\}, r_k \neq r_j$ and $\exists u \in r_k : (u-O \subset s-O)$. Since $r \in R_2$, it means that s was not deleted while creating the sets R' and R'', nor was it deleted while creating R_2. We will examine the consequences of each of these two cases:

a. Suppose that both r_j and r_k belong to one of $\{r_1,\ldots,r_i\}$ or $\{r_{i+1},\ldots,r_n\}$, say that they both belong to $\{r_1,\ldots,r_i\}$. Then, since s was not deleted while creating R' it means that $[u-(o_1 \cup \ldots \cup o_i)] \subset [s-(o_1 \cup \ldots \cup o_i)]$. Now, if u is not deleted by the application of μ which creates R', then both $u-O$ and $s-O$ will be considered while creating R_2 and $s-O$ will be deleted by the application of μ, which is a contradiction; if u is deleted while creating R' then $\exists r_q \in \{r_1,\ldots,r_i\}$ such that $\exists p \in r_q : p-(o_1 \cup \ldots \cup o_i) \subset u-(o_1 \cup \ldots \cup o_i)$, meaning that both $p-(o_1 \cup \ldots \cup o_i)$ and $s-(o_1 \cup \ldots \cup o_i)$ belong to R' and therefore s will be deleted by the application of μ which creates R_2, which is a contradiction.

b. Suppose that r_j and r_k do not both belong to one of $\{r_1,\ldots,r_i\}$ and $\{r_{i+1},\ldots,r_n\}$. Then s would be deleted by the application of μ which creates R_2, which is a contradiction.

Therefore $R_1=R_2$. □

Lemma 2: Given two supported wffs with the same OS, if their RS was obtained exclusively by successive applications of the μ operation then the supported wffs have the same RS as well.

Proof: It follows directly from Lemma 1. □

Theorem 5: In the knowledge base resulting from the application of the rules of inference of the SWM system, if two supported wffs have the same OS then they have the same RS as well.

Proof: The proof will be done by complete induction on the number of applications of rules of inference.

The only supported wffs which can be obtained by applying one rule of inference only are hypotheses and since the rule of hyp guarantees that their OS is unique the theorem is verified.

Suppose, by induction hypothesis, that all the supported wffs obtained by the application of less than n rules of inference verify the conditions of the theorem. We will have to show that the supported wff obtained by the application of the n-th rule of inference also satisfies the statement of the theorem. We will group the rules of inference of SWM according to the type of OS and RS they produce and will discuss the OS and RS of the supported wff produced by the application of such type of rule.

1. Creation of new OSs (hyp). The rule of hyp creates a supported wff with a new OS. The assumption behind the application of this rule is that there is no supported wff in the knowledge base with such OS and therefore this supported wff satisfies the conditions of the theorem.

2. Change in RS only (CRS). Upon detection of an inconsistent set, χ, this rule changes the RS of every supported wff whose OS is not disjoint from χ.

Every supported wff F | t,o,R such that $\chi \cap o \neq \emptyset$ is replaced by F | t,o,$\sigma(R \cup \{\chi - o\})$. If prior the application of this rule all the supported wffs with the same OS had the same RS the same condition will be verified after the application of the rule since the RS's of supported wffs with the same OS are affected in the same way. An important point to note is that $\sigma(R \cup \{\chi - o\})$ = $\mu(R \cup \{\chi - o\}, o)$ and thus the RS of the resulting supported wffs are the same as if the μ operation had been applied.

3. No change in OS nor RS ($\neg$E, $\wedge$I(part 1), $\wedge$E, $\vee$I, $\exists$I, $\exists$E). The application of one of these rules creates a new supported wff whose OS and RS are the same as the OS and RS of a previous supported wff, therefore, by the induction hypotheses the statement of the theorem is verified.

4. The OS is the union of the OSs of the parent wffs (MP, MT, $\wedge$I(part 2), $\vee$E, $\forall$E). Using one of these rules, if we combine $F_1 \mid t_1, o_1, r_1$ and $F_2 \mid t_2, o_2, r_2$ we obtain $F_3 \mid t_3, o_1 \cup o_2, \mu(\{r_1, r_2\}, \{o_1, o_2\})$. Lemma 2 guarantees that, if only μ operations had been applied to form the RSs of the supported wffs in the knowledge base, the RS of F_3 is the same as the RS of any wff whose OS is $o_1 \cup o_2$.

5. The resulting OS is one (or several) hypothesis short ($\rightarrow$I, $\neg$I, $\forall$I). Using one of these rules, we take one supported wff $F_1 \mid t_1, o, r$ and create a new supported wff $F_2 \mid t_2, o - \{H_1, \ldots, H_n\}, (o - \{H_1, \ldots, H_n\})$, recall that $(O) = \mu(\{r : \exists h \in O : r = \mathbf{rs}(h)\}, \{o : \exists h \in O : o = \mathbf{os}(h)\})$ and therefore Lemma 2 guarantees that, if only μ operations had been applied to form the RSs of the supported wffs in the knowledge base, the statement of the theorem is verified.

As a final remark it should be noticed that since the application of σ in (2) can be reduced to an application of μ, the RS of *every* supported wff in the knowledge base is created by successive applications of the μ operation and therefore if two supported wffs

in the knowledge base have the same OS they also have the same RS. □

Corollary 5.1: Every OS has recorded with it *every* known inconsistent set.

Theorem 6: Suppose that $C=\{H_1, \ldots, H_n\}$ is a context which is not known to be inconsistent. Then, for any two wffs, say A and B, in the BS defined by the context C, we have Combine(A,B)=true.

Proof: Suppose by way of contradiction that the wffs A : t_a, o_a, r_a and B : t_b, o_b, r_b belong to the BS defined by the context C and that Combine(A,B)=false. Since A and B belong to the BS defined by C we have $o_a \subset C$ and $o_b \subset C$. Since Combine(A,B)=false, one of the following conditions holds:

1. $\exists$ $r \in rs(A)$: $r \subset os(B)$: since $os(B) \subset \{H_1, \ldots, H_n\}$ we have that $r \subset \{H_1, \ldots, H_n\}$. By definition of restriction set we know that $(o_a \cup r) \vdash \rightarrow\leftarrow$. Since $o_a \subset \{H_1, \ldots, H_n\}$ and $r \subset \{H_1, \ldots, H_n\}$ we have that $(o_a \cup r) \subset \{H_1, \ldots, H_n\}$. Therefore, there exists a set $S \subset \{H_1, \ldots, H_n\}$ such that $S \vdash \rightarrow\leftarrow$. By Theorem 3 $\{H_1, \ldots, H_n\} \vdash \rightarrow\leftarrow$ which contradicts the assumption that C is not known to be inconsistent.
2. $\exists$ $r \in rs(B)$: $r \subset os(A)$: the same line of reasoning used in 1 will derive a contradiction.

Therefore Combine(A,B)=true. □

Corollary 6.1: If one uses a context which is not known to be inconsistent then the system using SWM does not need to check for combinability between the wffs before the application of rules of inference.

Appendix 2 - Relevance Logic

The point of this appendix is to introduce the terminology used by Anderson and Belnap in one of their relevance logic systems and to show how it is used to effectively block some of the results, obtainable in classical logic, which Anderson and Belnap consider to be irrelevant results. Anderson and Belnap's relevance logic was taken as the starting point for developing the SWM system. The main features of relevance logic used in the SWM system are the way it keeps track of which hypotheses were used in the derivation of a given wff and the way this is used to restrict the application of certain rules of inference.

Relevance logic was proposed by Anderson and Belnap [Anderson and Belnap 75], reacting against the lack of relevance in classical logic. Among other things, relevance logic challenges classical logic with respect to the classical concept of validity: Anderson and Belnap argue that if one proposition entails another, then there must be an element of causality that relevantly connects them, and, for that reason, they do not recognize as valid some of the arguments classified as valid by classical logic. In particular, they explicitly deny the so-called paradoxes of implication: $A \rightarrow (B \rightarrow A)$, anything implies a true proposition; and $(A \wedge \neg A) \rightarrow B$, a contradiction implies anything. To their (semantic) notion of entailment, there corresponds a (syntactic) notion of deducibility according to which B is deducible from A only if the derivation of B genuinely uses, and does not simply take a detour via, A.

We briefly describe how Anderson and Belnap define deducibility in a natural deduction system, the *FR system* [Anderson and Belnap 75, pp.346-348]. Most of this methodology was adopted in the SWM system. A natural deduction system, e.g. [Fitch 52], contains no axioms, only rules of inference. The rules of inference of a natural deduction system typically contain:

1. A rule of hypothesis which enables one to get started without the need of axioms from which to begin.
2. Two rules of inference for each logical symbol called the introduction and elimination rules. The introduction rule tells how to introduce an occurrence of the logical symbol

(logical symbols are either logical connectives or quantifiers) and is written σI, 'σ' being the logical symbol. The elimination rule tells how to eliminate an occurrence of the symbol and is written σE.

A proof is defined to be a nested set of subproofs. A subproof is a list of well-formed formulas (wffs) and/or subproofs. Each wff is contained in a subproof. Subproofs are initiated every time a new hypothesis is introduced (which can be done at any point) and terminated when the hypothesis is discharged. There is one outermost subproof, called "categorical", in which no hypotheses are assumed, and the remaining subproofs are called "hypothetical". Theorems are wffs in the categorical subproof.

In the FR system, to ensure that B is deducible from A only if A is *used* in the derivation of B, Anderson and Belnap restrict the classical rules of natural deduction, as follows:

1. Within a deduction, each wff is associated with a set containing references to all the hypotheses that were *really* used in its derivation. This corresponds to the *Origin Set* (OS) in the SWM system. We denote the fact that A is a wff with OS α by writing A,α.
2. The rules of inference are stated taking OSs into account, blocking what are considered to be irrelevant applications of the rules, which are allowed in classical logic.

In the FR system, whenever a new hypothesis is introduced, it is associated with a singleton OS whose element is an identifier that never appeared before in the proof[53]. The rules of inference of the FR system are stated so that all the wffs derived using a particular hypothesis will have its identifier in their OS. Theorems are wffs with empty OS. When a rule of inference is applied, the resulting wff is associated with an OS that is either the union of the OSs of the parent wffs, the OS of the parent wff(s), or the set difference of the OSs of the parent wffs. To give an idea of how the OSs can be formed, we will elaborate on two rules, →I and ∧I.

[53]Relevance logic systems typically use natural numbers as elements of the OSs.

The rule of →I states that if A is a hypothesis with OS {k}, B is a derived wff with OS $\alpha \cup \{k\}$ (meaning that A was genuinely used in the derivation of B), and they are both in the same subproof, then one can deduce A→B (in the subproof immediately containing the subproof initiated by the introduction of the hypothesis A) and associate this new wff with the OS α. This rule is schematically presented in Figure A2.1. Notice that A→B does not depend on hypothesis A. This is the reason for the set difference operation performed on the OS of B to obtain the OS of A→B.

```
m | A,{k}        Hyp
  |
  | . . .
  |
n | B,α∪{k}
  | A→B,α        →I(m,n)
```

Figure A2.1
FR system's →I

The rule of ∧I states that if A and B are wffs with the same OS, then one can deduce A∧B and associate it with that OS. This rule is represented in Figure A2.2.

```
    . . .
m   A,α
    . . .
n   B,α
    A∧B,α     ∧I (m,n)
```

Figure A2.2
FR system's ∧I

This rule may seem too strongly stated, but it must be so in order to restrict the gratuitous introduction of irrelevancies. Suppose that ∧I allowed the conjunction of wffs with different OSs, resulting in a wff whose OS was the union of the OSs of the parent wffs.

1	A,{1}	Hyp
2	B,{2}	Hyp
3	A,{1}	Reit (1)
4	B,{2}	Rep (2)
5	A∧B,{1,2}	∧I (3,4) ??
6	A,{1,2}	∧E (5)
7	B→A,{1}	MP (2,6)
8	A→(B→A),{}	MP (1,7)

Figure A2.3
"Proof" in the FR system

Figure A2.3 shows how we could introduce irrelevancies in the system. The application of ∧E to the wff in line 5, which resulted from such a use of ∧I, allows the hypothesis of line 2 to be "smuggled into" the OS of A (line 6), thereby allowing the "proof" of A→(B→A),one of the paradoxes of implication. The proof makes use of some rules of inference which have not been discussed, namely Ret, Rep, ∧E and MP (for a description of these rules refer to [Martins 83]).

SNEPS CONSIDERED AS A FULLY INTENSIONAL PROPOSITIONAL SEMANTIC NETWORK

Stuart C. Shapiro, William J. Rapaport

85-15 October 1985

Department of Computer Science
University at Buffalo (SUNY)
226 Bell Hall
Buffalo, New York 14260

The SNePS Semantic Network Processing System is a semantic network language with facilities for building semantic networks to represent virtually any kind of information, retrieving information from them, and performing inference with them. Users can interact with SNePS in a variety of interface languages, including a LISP-like user language, a menu-based screen-oriented editor, a graphics-oriented editor, a higher-order-logic language, and an extendible fragment of English.

This article discusses the syntax and semantics for SNePS considered as an intensional knowledge-representation system and provides examples of uses of SNePS for cognitive modeling, data-base management, pattern recognition, expert systems, belief revision, and computational linguistics.

This is a draft of a chapter forthcoming in Gorden McCalla and Nick Cercone, *Knowledge Representation* (Berlin: Springer-Verlag).

SNePS CONSIDERED AS A FULLY INTENSIONAL PROPOSITIONAL SEMANTIC NETWORK

Stuart C. Shapiro and William J. Rapaport

Department of Computer Science
University at Buffalo
State University of New York
Buffalo, NY 14260

1. INTRODUCTION.

A semantic network is a data structure typically consisting of labeled nodes and labeled, directed arcs. The SNePS *S*emantic *Ne*twork *P*rocessing *S*ystem (Shapiro 1979a) can be viewed as a semantic network language with facilities for (1) building semantic networks to represent virtually any kind of information or knowledge, (2) retrieving information from them, and (3) performing inference with them. (For a definition of SNePS as an abstract data type, see Morgado, forthcoming.) The user can interact with SNePS in a variety of interface languages, including: SNePSUL, a LISP-like SNePS User Language; SENECA, a menu-based, screen-oriented editor; GINSENG, a graphics-oriented editor; SNePSLOG, a higher-order-logic language (in the sense in which PROLOG is a first-order-logic language) (McKay and Martins 1981; Shapiro, McKay et al. 1981); and an extendible fragment of English, using an ATN parsing and generating grammar (Shapiro 1982).

SNePS is the descendent of SAMENLAQ (Shapiro, Woodmansee, and Kreuger 1968; Shapiro and Woodmansee 1969) and MENTAL (Shapiro 1971a, 1971b). It was developed with the help of the Semantic Network Research Group at Indiana University and at the University at Buffalo. The current version is implemented in Franz LISP and runs on VAX 11/750s and 780s in the Department of Computer Science at Buffalo. An earlier version was implemented in ALISP on a CDC Cyber 730, and there are installations at other universities in the U.S. and Europe.[1]

2. INTENSIONAL KNOWLEDGE REPRESENTATION.

SNePS can be used to represent propositions about entities in the world having properties and standing in relations. Roughly, nodes represent the propositions, entities, properties, and relations, while the arcs represent structural links between these. Thus, SNePS can be treated as a programming language whose main data structure is the semantic network (just as LISP's main data structure is the list), as a very powerful relational database management system with inferencing, or as an expert system shell. Examples of such uses of SNePS will be presented in Section 4.

The entities represented by the nodes in these uses will usually be *extensional*. Roughly, extensional entities are those whose "identity conditions" (the conditions for deciding when "two" of them are really the "same") do not depend on their manner of representation. Alternatively, they may be characterized as those entities satisfying the following rough principle:

> Two extensional entities are equivalent (for some purpose) if and only if they are identical (i.e., if and only if "they" are really one entity, not two).

[1] This research was supported in part by SUNY Buffalo Research Development Fund grant #150-9216-F (Rapaport) and in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008 (Shapiro). We wish to thank Michael Almeida, James Geller, João Martins, Jeannette Neal, Sargur N. Srihari, Jennifer Suchin, and Zhigang Xiang for supplying us with descriptions of their projects, and Randall R. Dipert and the members of SNeRG (the SNePS Research Group) for comments and discussion.

For example, the following are extensional:

the Fregean referent of an expression;
physical objects;
sentences;
truth values;
mathematical objects such as:
    sets,
    functions defined in terms of their input-output behavior (i.e., as sets of ordered pairs),
    n-*place relations* defined in terms of sets of ordered *n*-tuples.

SNePS nodes can also be used to represent mental entities. If used in this way, SNePS nodes must represent *intensional* entities. Roughly, intensional entities are those whose identity conditions *do* depend on their manner of representation. Alternatively, they are those entities that satisfy the following rough principle:[2]

> Two intensional entities might be equivalent (for some purpose) without being identical (i.e., they might really be two, not one).

For example, the following are intensional:

the Fregean sense of an expression;
concepts;
propositions;
properties;
algorithms;
objects of thought, including:
    fictional entities (such as Sherlock Holmes),
    non-existents (such as the golden mountain),
    impossible objects (such as the round square)

In some cases, the importance of being able to represent such things is clear; in other cases, perhaps less so. Using SNePS to represent extensional entities in the world does not preclude it from representing intensional entities. Indeed, we believe that it *must* represent intensional entities. Nor does using SNePS to represent the relations between a mind and the world preclude it from representing extensional entities. Indeed, to represent the *relations*, it would *have* to represent extensional entities (cf. Rapaport 1976, 1978; McCarthy 1979).

However, if SNePS is going to be used just to represent a mind—that is, a mind's model of the world—then *it does not need to represent any extensional objects*. SNePS can then be used either to model the mind of a particular cognitive agent or to build such a mind—i.e., to *be* a cognitive agent itself. For the sake of clarity, we refer to this agent as CASSIE (the *C*ognitive *A*gent of the *S*NePS *S*ystem—an *I*ntelligent *E*ntity).

There have been a number of arguments presented in both the AI and philosophical literature in the past few years for the need for intensional entities. Among them, the following considerations seem to us to be especially significant:

---

[2] Alternatively, intensional entities may be characterized as satisfying the following five criteria: (1) They are non substitutible in referentially opaque contexts. (2) They can be indeterminate with respect to some properties. (3) They need not exist. (4) They need not be possible. (5) They can be distinguished even if they are necessarily identical (e.g., *the sum of 2 and 2* and *the sum of 3 and 1* are distinct objects of thought).

(Principle of Fine-Grained Representation)

> The objects of thought (i.e., intentional objects) are intensional: a mind can have two or more objects of thought that correspond to only one extensional object.

To take the classic example, the Morning Star and the Evening Star might be distinct objects of thought, yet there is only one extensional object (viz., a certain astronomical body) corresponding to them.

(Principle of Displacement)

> Cognitive agents can think and talk about non-existents: a mind can have an object of thought that corresponds to no extensional object.

Again to take several classic examples, cognitive agents can think and talk about fictional objects such as Santa Claus, possible but non-existing objects such as a golden mountain, impossible objects such as a round square, and possible but not-yet-proven-to-exist objects such as theoretical entities (e.g., black holes). (For other arguments, see: Castañeda 1972; Woods 1975; Rapaport 1976, 1985a, and forthcoming; Brachman 1977; Routley 1979, cf. Rapaport 1984a; Parsons 1980, cf. Rapaport 1985b.)

If nodes only represent intensions (and extensional entities are not represented in the network), how do they link up to the external, extensional world? One answer is by means of a LEX arc (see (Syn.1) and (Sem.1), below): The nodes at the head of the LEX arc are *our* (the user's) interpretation of the node at its tail. The network without the LEX arcs and their head-nodes displays the *structure* of CASSIE's mind (*cf.* Carnap 1928, Sect. 14).

Another answer is by means of sensors and effectors, either linguistic or robotic. The robotic sort has been discussed in Maida and Shapiro 1982. Since so many AI understanding systems deal exclusively with language, here we consider a system with a keyboard as its sense organ and a CRT screen as its only effector.

Since the language system interacts with the outside world only through language, the only questions we can consider about the connections of its concepts with reality are questions such as:

> Does it use words as we do?
> When it uses word *w*, does it mean the same thing as when I use it?
> When I use word *w*, does it understand what I mean?

The perceptual system of the language system is its parser/analyzer—the programs that analyze typed utterances and build pieces of semantic network. The motor system is the generator—the programs that analyze a section of the semantic network and construct an utterance to be displayed on the CRT. One crucial requirement for an adequate connection with the world is simple consistency of input-output behavior. That is, a phrase that is analyzed to refer to a particular node should consistently refer to that node, at least while there is no change in the network. Similarly, if the system generates a certain phrase to describe the concept represented by a node, it should be capable of generating that same phrase for that same node, as long as nothing in the network changes. Notice that it is unreasonable to require that if a phrase is generated to describe a node, the analyzer should be able to find the node from the phrase: The system might know of several brown dogs and describe one as "a brown dog"; it could not be expected to find that node as the representation of "a brown dog" consistently.

If we are assured of the simple input-output consistency of the system, the main question left is whether it uses words to mean the same thing as we do. It is the same question that we would be concerned with if we were talking with a blind invalid, although in that case we would assume the answer was 'Yes' until the conversation grew so bizzare that we were forced to change our minds. As the system (or the invalid) uttered more and more sentences using a particular word or phrase, we would become more and more convinced that it meant what we would mean by it, or that it meant what we might have described with a different word or phrase ("Oh! When you say 'conceptual

dependency structure', you mean what I mean when I say 'semantic network'."), or else that we *didn't* know what was meant, or that it was not using it in a consistent, meaningful way (and hence that the system (or invalid) did not know what it was talking about). As long as the conversation proceeds without our getting into the latter situation, the system has all the connections with reality it needs.

In the next section, we offer our interpretation of a particular use of SNePS in terms of a philosophical theory of mental entities inspired by Alexius Meinong's Theory of Objects. Briefly, SNePS nodes will be taken as representing the objects of thought and SNePS arcs will be taken as representing the internal structuring of these objects with their external relations and properties.

3. DESCRIPTION OF SNePS.

Information is represented in SNePS by means of *nodes* and *arcs*. Since the meaning of a node is determined by what it is connected to in the network, there are no isolated nodes. Nodes that only have arcs pointing *to* them are considered to be unstructured or *atomic*. They include:

(A1) *sensory* nodes, which—when SNePS is being used to model a mind—represent interfaces with the external world (in the examples that follow, they will represent words, sounds, or utterances);

(A2) *base* nodes, which represent individual concepts and properties;

(A3) *variable* nodes, which represent arbitrary individuals (cf. Fine 1983) or arbitrary propositions.

Molecular nodes, which have arcs emanating *from* them, include:

(M1) *structured individual* nodes, which represent structured individual concepts or properties (i.e., concepts and properties represented in such a way that their internal structure is exhibited);

(M2) *structured proposition* nodes, which represent propositions; those with no incoming arcs represent *beliefs* of the system.[3] (Note that structured proposition nodes can also be considered to be structured individuals.) Proposition nodes are either *atomic* (representing atomic propositions) or are *rule nodes*. Rule nodes represent deduction rules and are used by SNIP, the *S*NePS *I*nference *P*ackage, for node-based deductive inference. (For details, see Shapiro 1977, 1978; McKay and Shapiro 1980, 1981; Shapiro and McKay 1980; Shapiro, Martins, and McKay 1982; and Martins and Shapiro 1983.)

For each of the three categories of molecular nodes (structured individuals, atomic propositions, and rules), there are *constant* nodes of that category and *pattern* nodes of that category representing arbitrary entities of that category. SNePS satisfies the following:

(Uniqueness Principle)

There is a one-to-one correspondence between nodes and represented concepts.[4]

This principle guarantees that nodes represent intensional objects.

There are a few built-in arc labels, used mostly for rule nodes. All other arc labels are defined by the user, typically at the beginning of an interaction with SNePS (though new labels can be defined at any time). *Paths* of arcs can also be defined, allowing for *path-based* inference, including property

[3] There is a need to distinguish between structured proposition nodes with no incoming arcs and structured individual nodes with no incoming arcs; the latter, of course, are not beliefs of the system. There is also a need to distinguish between beliefs of the system and those propositions that the system is merely contemplating or "assuming" temporarily (cf. Meinong 1910). We are currently adding this capability to SNePS by means of an *assertion* operator ('!').

[4] In Maida and Shapiro 1982: 291, this name was given to only half of the Principle as stated here: "each concept represented in the network is represented by a unique node".

inheritance within generalization hierarchies (see below; cf. Shapiro 1978, Srihari 1981, and Tranchell 1982).

3.1. CASSIE—A Model of a Mind.

Since most arcs are user-defined, perhaps the best way of explaining the syntax and semantics of SNePS is by example. Accordingly, we shall describe the way in which we have been using SNePS to build a model of the mind of the cognitive agent, CASSIE. Using Brachman's (1979) terminology, insofar as SNePS is a semantic network system at the logical level and can thus be used to define one at the epistemological or conceptual level, CASSIE is SNePS being used at a conceptual level.

As described in Maida and Shapiro 1982 and Rapaport 1985a and forthcoming, the nodes represent the objects of CASSIE's thoughts—the things she thinks about, the properties and relations with which she characterizes them, her beliefs, her judgments, etc. It is an important fact, expressed by the Principle of Displacement, that a cognitive agent is able to think about virtually anything, including fictional objects, possible but non-existing objects, and impossible objects. Thus, any theory that would account for this fact must be embedded in a non-standard logic, and its semantics cannot be limited to merely *possible* worlds. Theories based on the Theory of Objects of the turn-of-the-century Austrian philosopher-psychologist Alexius Meinong are of precisely this kind. (For details, see: Meinong 1904; Findlay 1963; Rapaport 1976, 1978, 1981, 1982; Castañeda 1972, 1975abc, 1977, 1979, and Tomberlin 1984; Routley 1979, cf. Rapaport 1984a; Parsons 1980, cf. Rapaport 1985b; Lambert 1983, cf. Rapaport 1984c; Zalta 1983.)

For present purposes, it will be enough to say that Meinong held that psychological experiences consist (in part) of a psychological *act* (such as thinking, believing, judging, wishing, etc.) and the *object* to which the act is directed (e.g., the object that is thought about or the proposition that is believed). Two kinds of Meinongian objects of thought are relevant for us:

(1) The *objectum*, or object of "simple" thoughts: Santa Claus is the objectum of John's act of thinking of Santa Claus. Objecta are the meanings of noun phrases.

(2) The *objective*, or object of belief, knowledge, etc.: that Santa Claus is thin is the objective of John's act of believing that Santa Claus is thin. Objectives are the meanings of sentences and other sentential structures; they may be thought of as propositions.

It is important to note that objecta need not exist and that objectives need not be true.

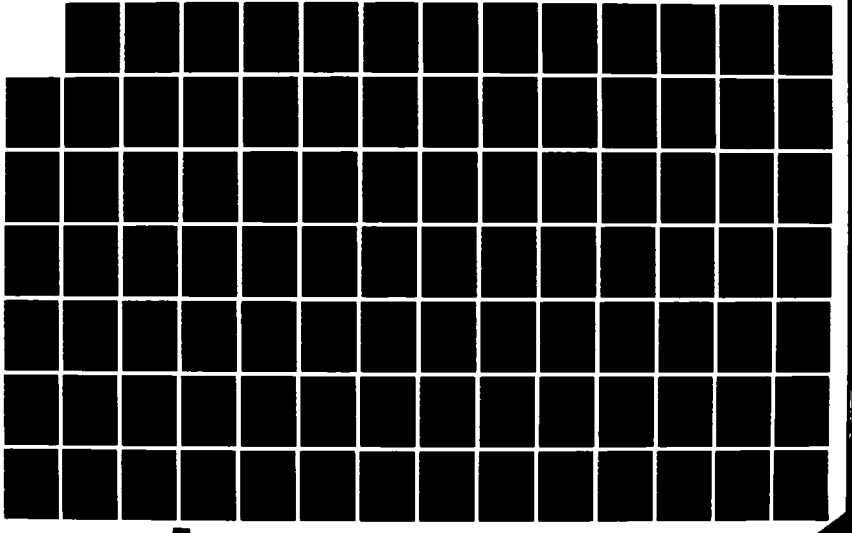
We apologize for this somewhat arcane terminology. But we want to distinguish between the objects of such acts as believing and the objects of such acts as thinking-of (or the components of such acts as believing); the actual arc labels are irrelevant. Moreover, it is important to note that not only are all represented things intensional, but that they are all objects of CASSIE's mental acts; i.e., they are all in CASSIE's mind (her "belief space")—they are all intentional. Thus, even if CASSIE represents the beliefs of someone else (e.g., John's belief that Lucy is rich, as in the conversation in the next section), the objects that she represents as being in that person's mind (as being in his "belief space") are actually CASSIE's representations of those objects—i.e., they are in CASSIE's mind.

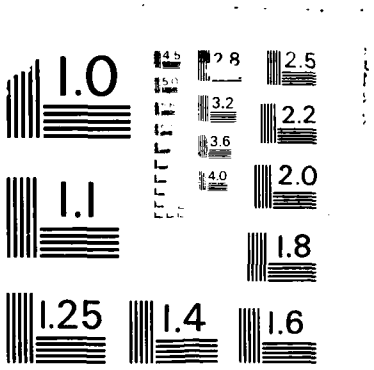
3.2. A Conversation with CASSIE.

Before presenting the arc-labels that we employ in representing CASSIE's "mind", it will prove useful to consider a typical interaction with her. An ATN parser/generator (Shapiro 1982) is being used to parse the English input into SNePS, and SNePS structures into English. (User input is on the lines beginning with the *-prompt; CASSIE's output is on the lines that follow.)

```
* (: young Lucy petted a yellow dog)
I understand that young Lucy petted a yellow dog
exec: 3.70 sec    gc: 0.00 sec
```

AD-A189 773 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC) 3/5
REVIEW OF TECHNICAL T.. (U) NORTHEAST ARTIFICIAL
INTELLIGENCE CONSORTIUM SYRACUSE NY J F ALLEN ET AL.
UNCLASSIFIED JUL 87 RADC-TR-86-210-V2-PT1 F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963

* (: what is yellow)
a dog is yellow
exec: 2.06 sec gc: 0.00 sec

* (: dogs are animals)
I understand that dogs are animals
exec: 1.63 sec gc: 0.00 sec

* (: who petted an animal)
young Lucy petted a yellow dog
exec: 3.90 sec gc: 0.00 sec

* (: Lucy is sweet)
I understand that young Lucy is sweet
exec: 2.26 sec gc: 0.00 sec

* (: Lucy is a girl)
I understand that sweet young Lucy is a girl
exec: 2.41 sec gc: 0.00 sec

* (: girls are people)
I understand that girls are people
exec: 1.61 sec gc: 0.00 sec

* (: what was petted by a person)
a yellow dog was petted by sweet young Lucy
exec: 4.26 sec gc: 0.00 sec

* (: the dog is named Rover)
I understand that Rover is a yellow dog
exec: 2.40 sec gc: 3.25 sec

* (: what was petted by a person)
yellow Rover was petted by sweet young Lucy
exec: 4.31 sec gc: 0.00 sec

* (: John believes that Lucy is rich)
I understand that John believes that Lucy is rich
exec: 4.70 sec gc: 0.00 sec

* (: who is rich)
I don't know.
exec: 1.58 sec gc: 0.00 sec

* (: who is sweet)
young Lucy is sweet
exec: 2.10 sec gc: 0.00 sec

* (: John believes that Lucy is old)
I understand that John believes that rich Lucy is old
exec: 4.71 sec gc: 0.00 sec

* (: John is a boy)
I understand that John is a boy
exec: 1.95 sec gc: 3.46 sec

```
* (: boys are people)
I understand that boys are people
exec: 1.60 sec    gc: 0.00 sec

* (: dogs are pets)
I understand that dogs are pets
exec: 1.63 sec    gc: 0.00 sec

* (: for every p and d if p is a person and d is a pet then p loves d)
I understand that for every d and p, if p is a person
and
  d is a pet
  then p loves d
exec: 6.13 sec    gc: 0.00 sec

* (: who loves a pet)
sweet young Lucy loves yellow Rover
and
John loves yellow Rover
exec: 8.86 sec    gc: 3.68 sec
```

3.3. Syntax and Semantics of SNePS.

In this section, we give a more formal presentation of the nodes and arcs used in this interaction, together with some other important ones. (We return to a more detailed examination of the interaction in the next section.) What we present here is our current model; we make no claims to completeness of the representational scheme. In particular, we leave for another paper a discussion of such structured individuals as the golden mountain or the round square, which raise difficult and important problems with predication and existence (for a discussion of these issues, see Rapaport 1985a and forthcoming). We begin with a few definitions.[5]

(Definition 1)

> A node *dominates* another node if there is a path of directed arcs from the first node to the second node.

(Definition 2)

> A *pattern* node is a node that dominates a variable node.

(Definition 3)

> An *individual* node is either a base node, a variable node, or a structured constant or pattern individual node.

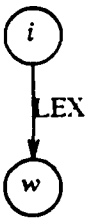
(Definition 4)

> A *proposition* node is either a structured proposition node or an atomic variable node representing an arbitrary proposition.

---

[5] These are actually only rough definitions; the interested reader is referred to Shapiro 1979a, Sect. 2.1, for more precise ones.

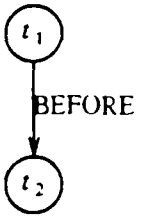
(Syn.1) If "w" is a(n English) word and "i" is an identifier not previously used, then



is a network, w is a sensory node, and i is a structured individual node.

(Sem.1) i is the Meinongian objectum corresponding to the utterance of w.

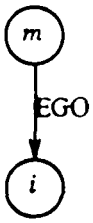
(Syn.2) If either "t_1" and "t_2" are identifiers not previously used, or "t_1" is an identifier not previously used and t_2 is a temporal node, then



is a network and t_1 and t_2 are *temporal* nodes, i.e. individual nodes representing times.

(Sem.2) t_1 and t_2 are Meinongian objecta corresponding to two times, the former occurring before the latter. (We do not distinguish time points from time intervals, because we feel that, conceptually, a time point is just an interval during which nothing happens, so whether a temporal node represents a point or an interval depends on what is attached to the node in the network; cf. Almeida, forthcoming.)

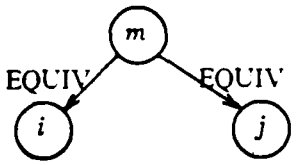
(Syn.3) If i is an individual node, and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.3) m is the Meinongian objective corresponding to the proposition that i is CASSIE's self-concept. (i would be expressed by CASSIE as "I".)

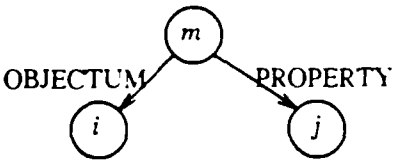
(Syn.4) If i and j are individual nodes, and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.4) m is the Meinongian objective corresponding to the proposition that Meinongian objecta i and j (are believed by CASSIE to) correspond to the same actual object. (Cf. Rapaport 1978, 1984b and Castañeda 1972, 1975b for analyses of this sort of relation, and Maida and Shapiro 1982 for a discussion of its use.)

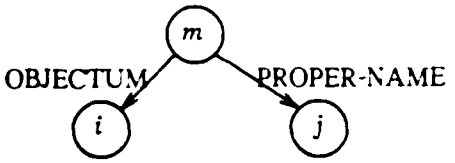
(Syn.5) If i and j are individual nodes and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.5) m is the Meinongian objective corresponding to the proposition that i has the property j.

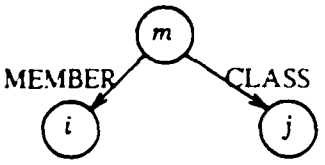
(Syn.6) If i and j are individual nodes and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.6) m is the Meinongian objective corresponding to the proposition that Meinongian objectum i's proper name is j. (j is the Meinongian objectum that is i's proper name; its expression in English is represented by a node at the head of a LEX-arc emanating from j.)

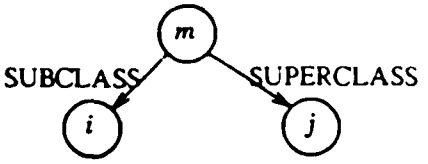
(Syn.7) If i and j are individual nodes and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.7) m is the Meinongian objective corresponding to the proposition that i is a (member of class) j.

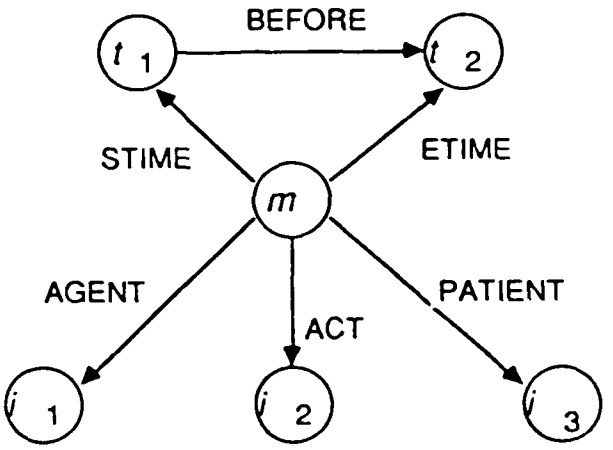
(Syn.8) If i and j are individual nodes and "m" is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.8) m is the Meinongian objective corresponding to the proposition that (the class of) is are (a subclass of the class of) js.

(Syn.9) If i_1, i_2, i_3 are individual nodes, t_1, t_2 are temporal nodes, and "m" is an identifier not previously used, then

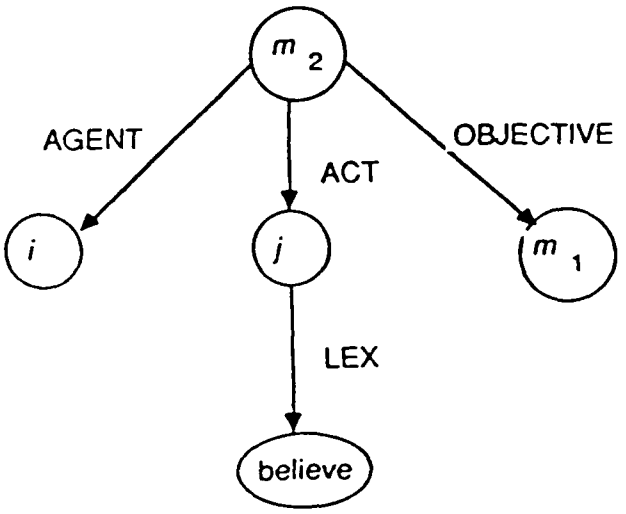


is a network and m is a structured proposition node.

(Sem.9) m is the Meinongian objective corresponding to the proposition that agent i_1 performs act i_2 to or on i_3 starting at time t_1 and ending at time t_2, where t_1 is before t_2.

It should be noted that the ETIME and STIME arcs are optional and can be part of any proposition node. (They are a provisional technique for handling tensed verbs; our current research on temporal representation is much more complex and is discussed in Section 4.7, below.)

(Syn.10) If m_1 is a proposition node, i is an individual node, j is the (structured individual) node with a LEX arc to the node, believe, and "m_2" is an identifier not previously used, then

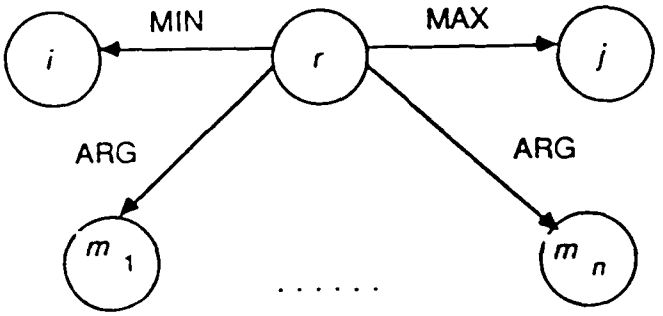


is a network and m_2 is a structured proposition node.

(Sem.10) m_2 is the Meinongian objective corresponding to the proposition that agent i believes proposition m_1.

Three special cases of (Syn.10) that are of interest concern *de re*, *de dicto*, and *de se* beliefs; they are illustrated in Figures 1-3. (For details, see Rapaport and Shapiro 1984, Rapaport 1984b.)

(Syn.11) If $m_1, \ldots, m_n$ are proposition nodes ($n \geq 0$), "i" and "j" are integers between 0 and n, inclusive, and "r" is an identifier not previously used, then



is a network, and r is a rule node.

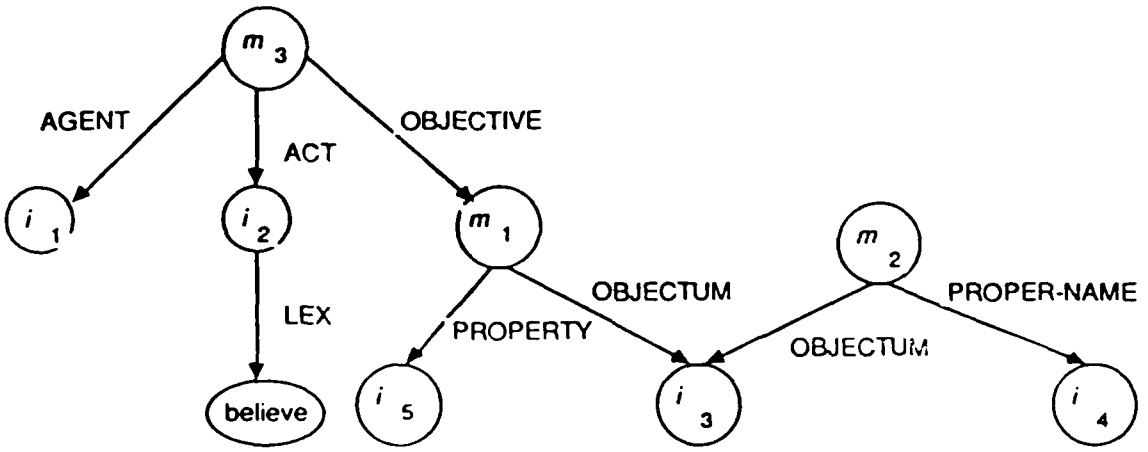


Figure 1. m_3 is the Meinongian objective corresponding to the proposition that agent i_1 believes *de re* of objectum i_3 (who is believed by CASSIE to be named i_4) that it has the property i_5.

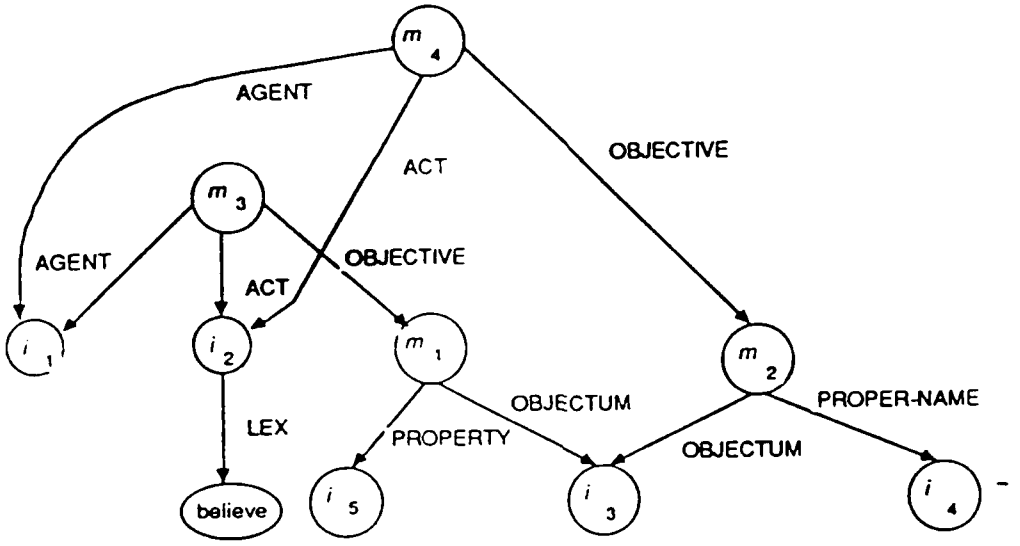


Figure 2. m_4 is the Meinongian objective corresponding to the proposition that agent i_1 believes *de dicto* that objectum i_3 (who is believed by i_1 to be named i_4) has the property i_5.

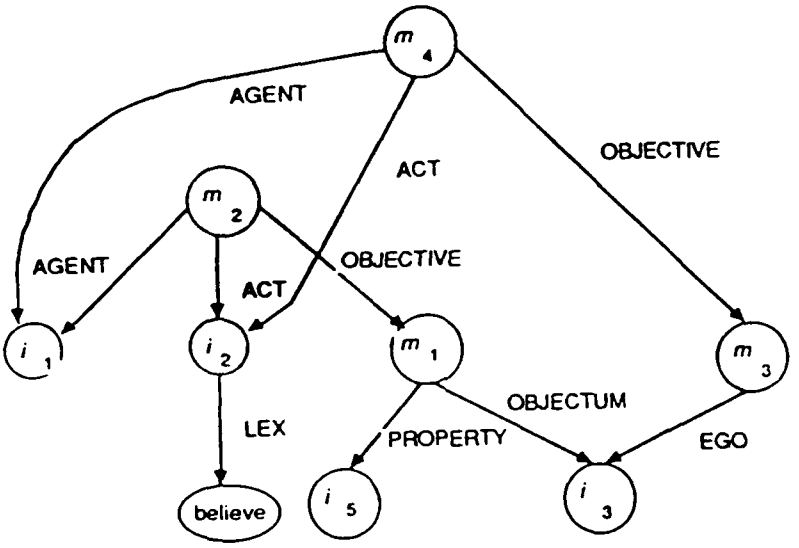
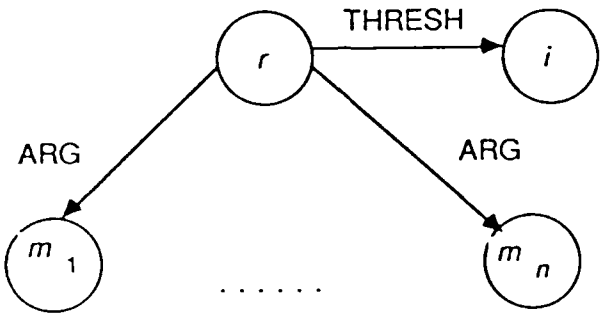


Figure 3. m_4 is the Meinongian objective corresponding to the proposition that agent i_1 believes *de dicto* that he himself (or she herself) has the property i_5.

(Sem.11) r is the Meinongian objective corresponding to the proposition that there is a relevant connection between propositions $m_1, \ldots, m_n$ such that at least i and at most j of them are simultaneously true.

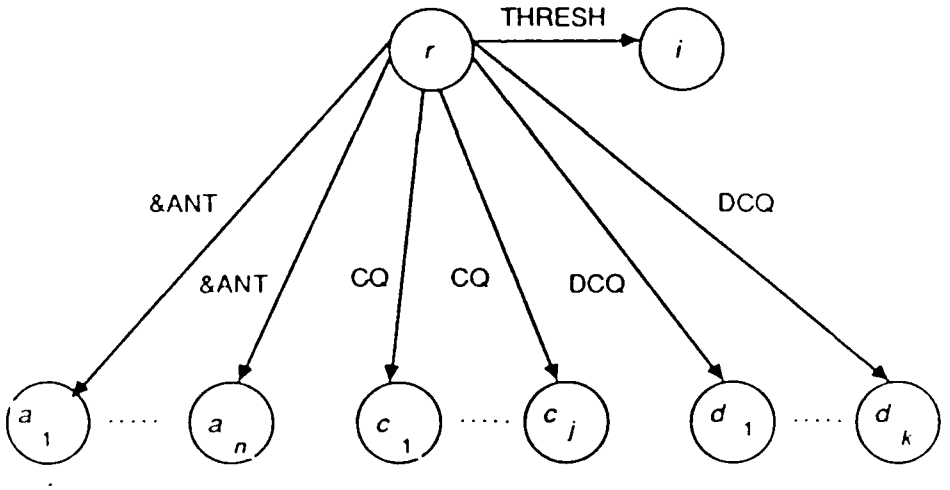
(Syn.12) If $m_1, \ldots, m_n$ are proposition nodes ($n \leq 0$), "i" is an integer between 0 and n, inclusive, and "r" is an identifier not previously used, then



is a network, and r is a rule node.

(Sem.12) r is the Meinongian objective corresponding to the proposition that there is a relevant connection between propositions $m_1, \ldots, m_n$ such that either fewer than i of them are true or they all are true.

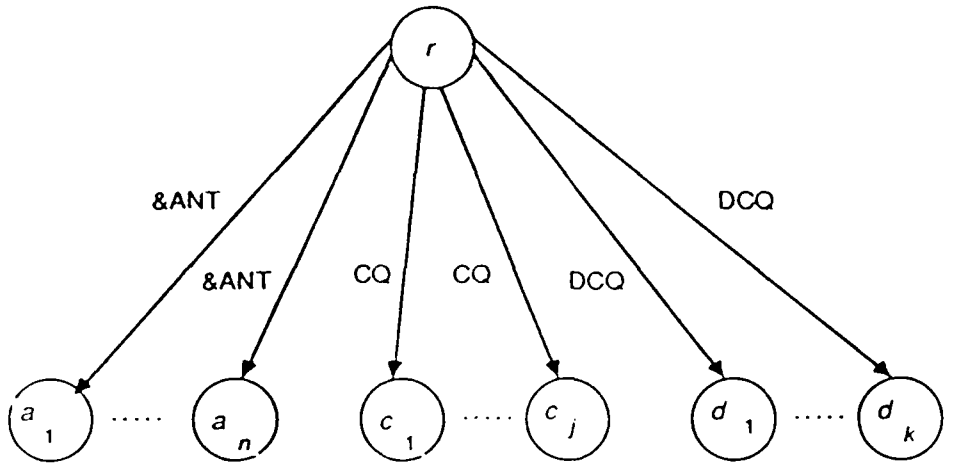
(Syn.13) If $a_1, \ldots, a_n, c_1, \ldots, c_j$, and $d_1, \ldots, d_k$ are proposition nodes ($n \geq 1$; $j, k \geq 0$; $j + k \geq 1$), "i" is an integer between 1 and n, inclusive, and "r" is an identifier not previously used, then



is a network, and r is a rule node.

(Sem.13) r is the Meinongian objective corresponding to the proposition that the conjunction of any i of the propositions $a_1, \ldots, a_n$ relevantly implies each c_l $(1 \leq l \leq j)$ and relevantly implies each d_l $(1 \leq l \leq k)$ for which there is not a better reason to believe it is false.

(Syn.14) If $a_1, \ldots, a_n, c_1, \ldots, c_j$, and $d_1, \ldots, d_k$ are proposition nodes $(n, j, k \geq 0)$, and "r" is an identifier not previously used, then

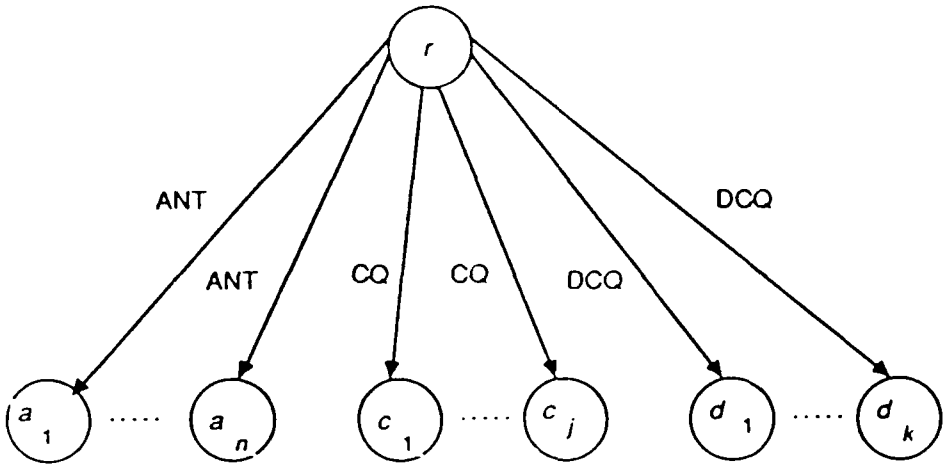


is a network, and r is a rule node.

(Sem.14) r is the Meinongian objective corresponding to the proposition that the conjunction of the propositions $a_1, \ldots, a_n$ relevantly implies each c_l $(1 \leq l \leq j)$ and relevantly implies each d_l $(1 \leq l \leq k)$ for which there is not a better reason to believe it is false.

The d_l are *default* consequences, in the sense that each is implied only if it is neither the case that CASSIE already believes *not* d_l nor that *not* d_i follows from non-default rules.

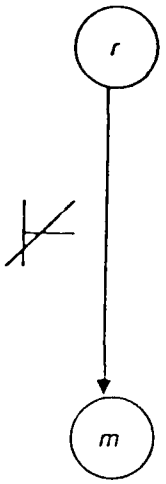
(Syn.15) If $a_1, \ldots, a_n, c_1, \ldots, c_j$, and $d_1, \ldots, d_k$ are proposition nodes $(n \geq 1;\ j, k \geq 0;\ j + k \geq 1)$, and "$r$" is an identifier not previously used, then



is a network, and r is a rule node.

(Sem.15) r is the Meinongian objective corresponding to the proposition that any a_i, $1 \leq i \leq n$, relevantly implies each c_l $(1 \leq l \leq j)$ and relevantly implies each d_l $(1 \leq l \leq k)$ for which there is not a better reason to believe it is false.

(Syn.16) If m is a proposition node, and "r" is an identifier not previously used, then



is a network, and r is a rule node.

(Sem.16) r is the Meinongian objective corresponding to the proposition that there is no good reason for believing proposition m.

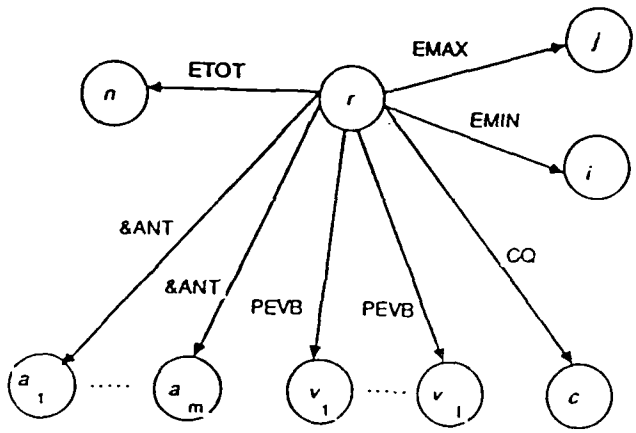
(Syn.17) If r is a rule node as specified by (Syn.9)-(Syn.16), and r dominates variable nodes $v_1, \ldots, v_n$, and, in addition, arcs labeled "AVB" go from r to each v_i, then r is a quantified rule node.

(Sem.17) r is the Meinongian objective corresponding to the proposition that the rule that would be expressed by r without the AVB arcs holds after replacing each v_i by any Meinongian object in its range.

(Syn.18) If r is a rule node as specified by (Syn.9)-(Syn.16), and r dominates variable nodes $v_1, \ldots, v_n$, and, in addition, arcs labeled "EVB" go from r to each v_i, then r is a quantified rule node.

(Sem.18) r is the Meinongian objective corresponding to the proposition that the rule that would be expressed by r without the EVB arcs holds after replacing each v_i by some Meinongian object in its range.

(Syn.19) If $a_1, \ldots, a_m$ and c are proposition nodes; $v_1, \ldots, v_l$ are variable nodes dominated by one or more of $a_1, \ldots, a_m, c$; "i", "j", and "n" are integers ($0 \leq i \leq j \leq n$); and "r" is an identifier not previously used; then



is a network, and r is a rule node.

(Sem.19) r is the Meinongian objective corresponding to the proposition that, of the n sequences of Meinongian objects which, when substituted for the sequence $v_1, \ldots, v_l$, make all the a_i believed propositions, between i and j of them also satisfy c. (For further details on such numerical quantifiers, see Shapiro 1979b.)

3.4. The Conversation with CASSIE, Revisited.

In this section, we shall review the conversation we had with CASSIE, showing the network structure as it is built—i.e., showing the structure of CASSIE's mind as she is given information and as she infers new information. (Comments are preceded by a dash.)

* (: young Lucy petted a yellow dog)
I understand that young Lucy petted a yellow dog

— CASSIE is told something, which she now believes. Her entire belief structure is shown in Figure 4a. The node labeled "now" represents the current time, so the petting is clearly represented as being in the past. CASSIE's response is "I understand that" appended to her English description of the proposition just entered.

* (: what is yellow)
a dog is yellow

— This response shows that CASSIE actually has some beliefs; she did not just parrot back the above sentence.

* (: dogs are animals)
I understand that dogs are animals

— CASSIE is told a small section of a class hierarchy.

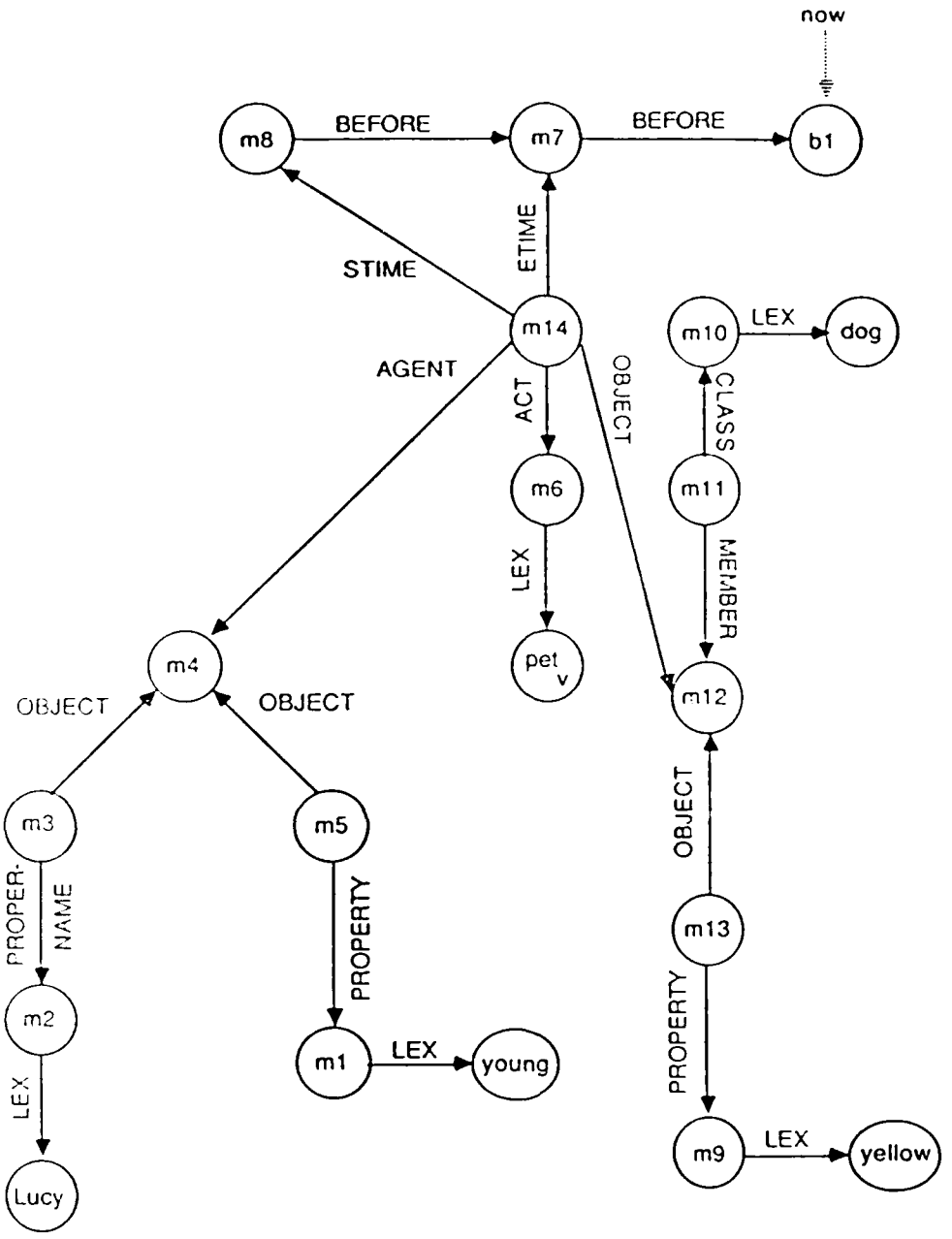


Figure 4a. Fragment of CASSIE's belief structure after being told that young Lucy petted a yellow dog.

* (: who petted an animal)
young Lucy petted a yellow dog

— CASSIE can answer the question using the class hierarchy, because, prior to the conversation, the inheritance rule

(def-path class (compose class (kstar (compose subclass- superclass))))

was given to SNePS. This rule says that the CLASS arc is implied by the path consisting of a class arc followed by zero or more occurrences of the two-arc path consisting of the converse SUBCLASS arc followed by the SUPERCLASS arc (see Shapiro 1978, Srihari 1981). The dog was called "a yellow dog" rather than "a yellow animal" because the redundant CLASS arc is not built. Figure 4b shows the current state of CASSIE's belief structure about the dog's classification and color.

* (: Lucy is sweet)
I understand that young Lucy is sweet

— CASSIE's response shows that she identifies this Lucy with the previous Lucy.

* (: Lucy is a girl)
I understand that sweet young Lucy is a girl

— The beginning of a class hierarchy for Lucy. Notice that all the adjectival properties of Lucy are mentioned.

* (: girls are people)
I understand that girls are people

— More of the class hierarchy is given.

* (: what was petted by a person)
a yellow dog was petted by sweet young Lucy

— Again, the proposition is retrieved using the CLASS inheritance rule.

* (: the dog is named Rover)
I understand that Rover is a yellow dog

— 'The dog' refers to the only dog CASSIE knows about, who is now given a name.

* (: what was petted by a person)
yellow Rover was petted by sweet young Lucy

— This is exactly the same question that was asked before. It is answered differently this time, because the dog now has a name, and CASSIE prefers to describe an individual by its name when it has one.

* (: John believes that Lucy is rich)
I understand that John believes that Lucy is rich

— At this point in our development of CASSIE, she interprets 'believes that' contexts to be *de dicto*, so she assumes that the Lucy that John has in mind is a different one from the Lucy that she knows. Figure 4c shows CASSIE's beliefs about the two Lucies.

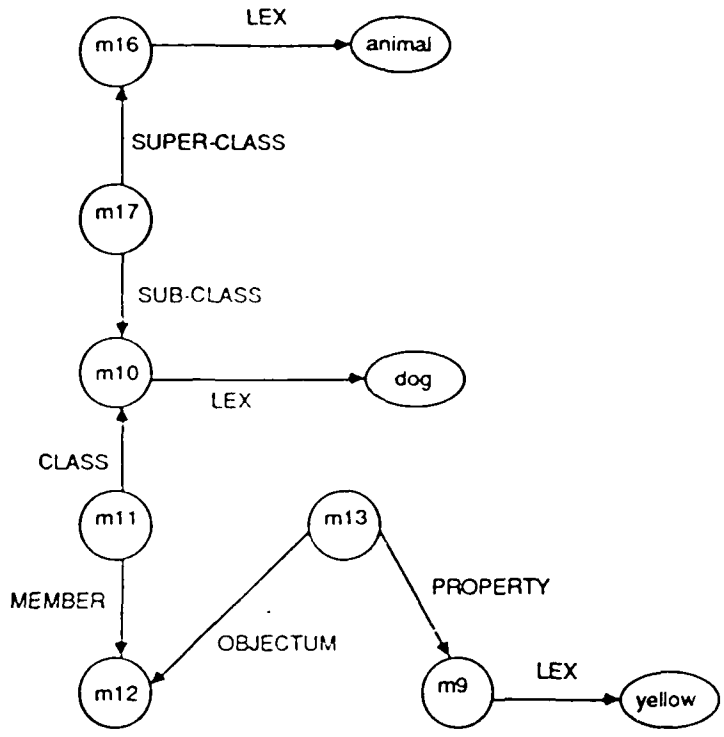


Figure 4b. CASSIE's belief structure about the dog's classification and color. (Node m12 represents the dog.)

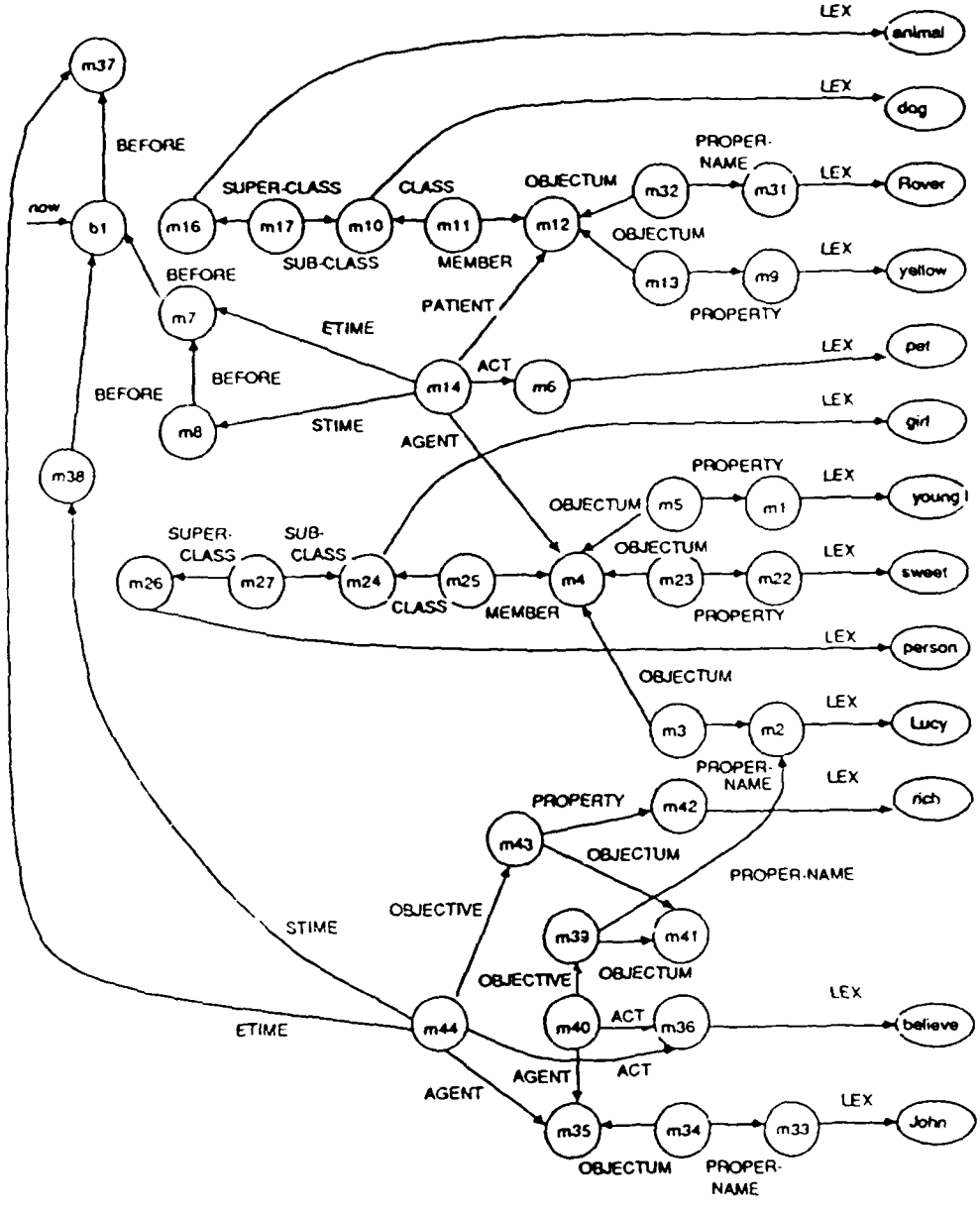


Figure 4c. A fragment of the network after CASSIE is told that John believes that Lucy is rich, showing CASSIE's beliefs about the two Lucies.

* (: who is rich)
I don't know.

— CASSIE knows no one who is rich. She only believes that *John* believes that someone (whom he believes to be named 'Lucy') is rich. The answer is 'I don't know', rather than 'no one is rich', because CASSIE doesn't use the closed-world hypothesis.

* (: who is sweet)
young Lucy is sweet

— This question is asked merely to demonstrate that Lucy is able to answer a "who is <property>" question when she has relevant beliefs.

* (: John believes that Lucy is old)
I understand that John believes that rich Lucy is old

— Even though CASSIE assumes that John knows a different Lucy than she knows, she assumes that all John's beliefs about "Lucy" are about the same Lucy.

* (: John is a boy)
I understand that John is a boy

— This and the next two inputs are given to establish more of the class hierarchy and to make it clear that when CASSIE answers the last question of this session, she is doing both path-based reasoning and node-based reasoning at the same time.

* (: boys are people)
I understand that boys are people

* (: dogs are pets)
I understand that dogs are pets

* (: for every p and d if p is a person and d is a pet then p loves d)
I understand that for every d and p, if p is a person
and
  d is a pet
  then p loves d

— Figure 4d shows how this node-based rule fits into the class hierarchy. This is, we believe, equivalent to the integrated TBox/ABox mechanism proposed for KRYPTON (Brachman et al. 1983, Brachman et al. 1985).

* (: who loves a pet)
sweet young Lucy loves yellow Rover
and
John loves yellow Rover

— The question was answered using path-based inferencing to deduce that Lucy and John are people and that Rover is a pet, and node-based inferencing to conclude that, therefore, Lucy and John love Rover.

The full network showing CASSIE's state of mind at the end of the conversation is given in Figure 4e.

4. EXTENSIONS AND APPLICATIONS OF SNePS.

In this essay, we have been advocating the use and interpretation of SNePS networks to model (the beliefs of) a cognitive agent. SNePS, however, is of much wider and more general applicability. In

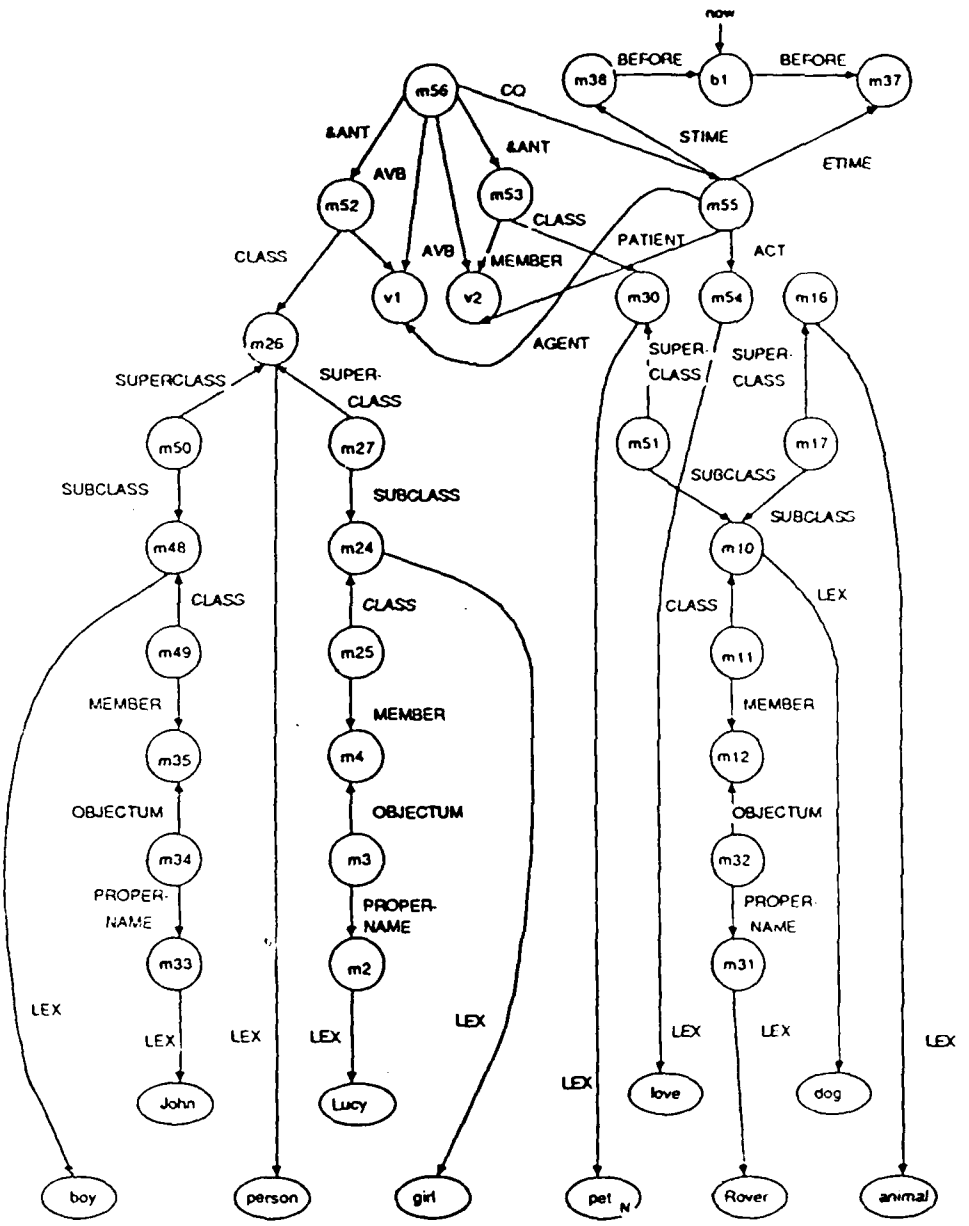


Figure 4d. A node-based rule in a class hierarchy.

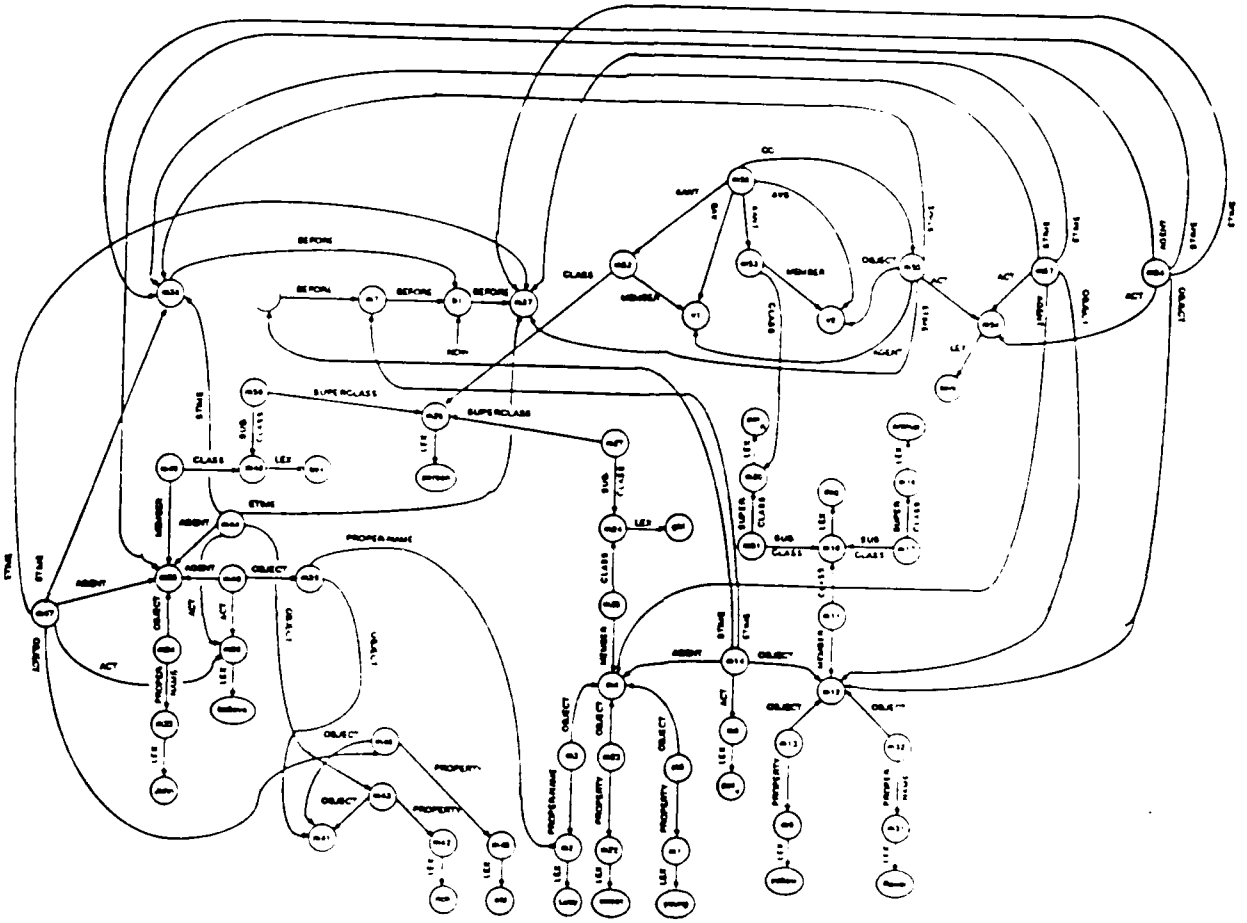


Figure 4e. CASSIE's beliefs at the end of the conversation.

this section, we give examples of recent and current research projects using SNePS in belief-revision, as a data-base management system, for developing several expert systems, and for representing temporal information in narratives. Even though most of these uses of SNePS do not explicitly involve a cognitive agent, nevertheless, in each case the asserted nodes can be treated as "beliefs" of the system: beliefs about the data base, beliefs about the various domains of the expert systems, beliefs about linguistics, etc.

4.1. SNePS as a Data-Base Management System.

SNePS can be used as a network version of a relational database in which every element of the relational database is represented by an atomic node, each row of each relation is represented by a molecular node, and each column label (attribute) is represented by an arc label. Whenever a row r has an element e in column c, the molecular node representing r has an arc labeled c pointing to the atomic node representing e. Relations (tables) may be distinguished by either of two techniques, depending on the particular relations and attributes in the relational database. If each relation has an attribute that does not occur in any other relation, then the presence of an arc labeled with that attribute determines the relationship represented by the molecular node. A review of the syntax of the CASSIE networks will show that this technique is used there. The other technique is to give every molecular node an additional arc (perhaps labeled "RELATION") pointing to an atomic node whose identifier is the name of the relation. Tables 1-4 show the Supplier-Part-Project database of Date (1981: 114). Notice that the SNAME and STATUS attributes only occur in the SUPPLIER relation; PNAME, COLOR, and WEIGHT only occur in the PART relation; JNAME only occurs in the PROJECT relation; and QTY only occurs in the SPJ relation. Figure 5 shows the SNePS network for part of this database.

Many database retrieval requests may be formulated using the **find** command of SNePSUL, the SNePS User's Language. The syntax of **find** is (**find** r_1 n_1 ... r_m n_m), where r_i is either an arc or a path, and n_i is either a node or a set of nodes (possibly the value of a nested call to **find**). The value of a call to **find** is the set of all nodes in the network with an r_1 arc to any node in the set n_1, an r_2 arc to any node in the set n_2, ..., and an r_m arc to any node in the set n_m. Free variables are prefixed by "?". An infix "-" between **find**s represents the set difference operator.

The session below shows some of the queries from Date (1981: 141-2) translated into **find** commands, and the results on the database shown above. (In each interaction, comments are preceded by semicolons, user input follows the '*'-prompt, and SNePS responses are on succeeding lines.)

```
; Get full details of all projects in London.
* (dump (find jname ?x city London))
(m18 (city (London)) (jname (tape)) (jnum (j7)))
(m16 (city (London)) (jname (collator)) (jnum (j5)))
(dumped)
exec: 0.10 sec     gc: 0.00 sec

; Get SNUM values for suppliers who supply project J1 with part P1
* (find snum  (find jnum j1 pnum p1))
(s1)
exec: 0.06 sec     gc: 0.00 sec

; Get JNAME values for projects supplied by supplier S1.
* (find (jname  jnum jnum  snum) s1)
(console sorter)
exec: 0.10 sec     gc: 0.00 sec
```

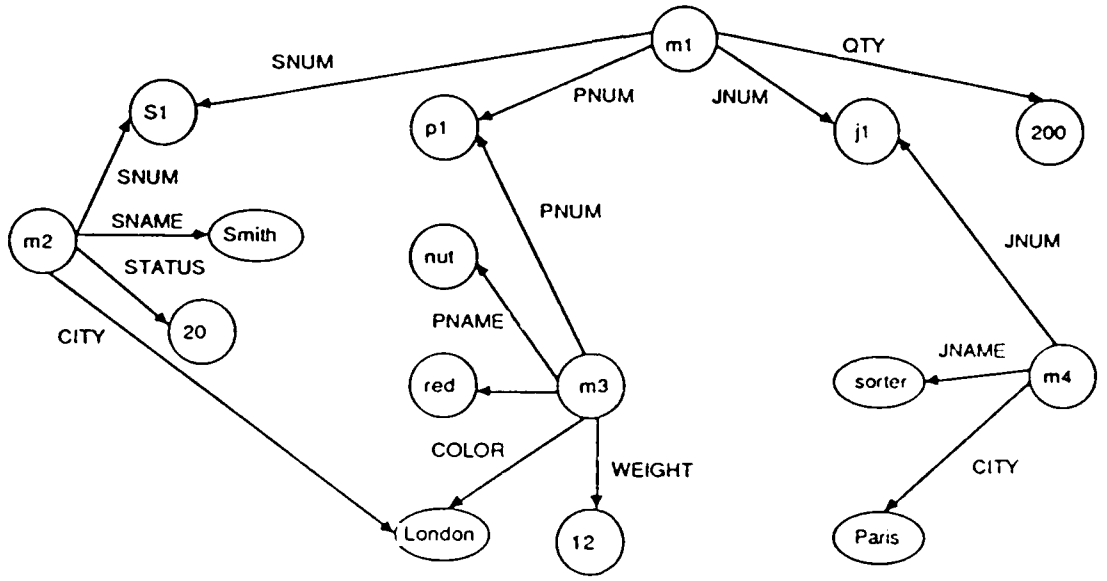


Figure 5. Fragment of SNePS network for the Supplier-Part-Project database.

Table 1: SUPPLIER

S#	SNAME	STATUS	CITY
s1	Smith	20	London
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	London
s5	Adams	30	Athens

Table 2: PART

P#	PNAME	COLOR	WEIGHT	CITY
p1	nut	red	12	London
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	London
p5	cam	blue	12	Paris
p6	cog	red	19	London

Table 3: PROJECT

J#	JNAME	CITY
j1	sorter	Paris
j2	punch	Rome
j3	reader	Athens
j4	console	Athens
j5	collator	London
j6	terminal	Oslo
j7	tape	London

Table 4: SPJ

S#	P#	J#	QTY
s1	p1	j1	200
s1	p1	j4	700
s2	p3	j1	400
s2	p3	j2	200
s2	p3	j3	200
s2	p3	j4	500
s2	p3	j5	600
s2	p3	j6	400
s2	p3	j7	800
s2	p5	j2	100
s3	p3	j1	200
s3	p4	j2	500
s4	p6	j3	300
s4	p6	j7	300
s5	p2	j2	200
s5	p2	j4	100
s5	p5	j5	500
s5	p5	j7	100
s5	p6	j2	200
s5	p1	j4	1000
s5	p3	j4	1200
s5	p4	j4	800
s5	p5	j4	400
s5	p6	j4	500

```
; Get S# values for suppliers who supply both projects J1 and J2.
* (find (snum- jnum) j1 (snum- jnum) j2)
(s3 s2)
exec: 0.08 sec     gc: 0.00 sec

; Get the names of the suppliers who supply project J1 with a red part.
* (find (sname- snum snum-)  (find jnum j1 (pnum pnum- color) red))
(Smith)
exec: 0.13 sec     gc: 0.00 sec

; Get S# values for suppliers who supply a London or Paris
;       project with a red part.
* (find snum- (find (jnum jnum- city) (London Paris)
                    (pnum pnum- color) red))
(s4 s1)
exec: 0.21 sec     gc: 0.00 sec

; Get P# values for parts supplied to any project by
;       a supplier in the same city.
* (find pnum- (find (jnum jnum- city) ?city (snum snum- city) ?city))
(p5 p4 p1 p2 p6 p3)
exec: 1.11 sec     gc: 0.00 sec
```

```
; Get J# values for projects not supplied with any red part
;       by any London supplier.
* ((find jnum- ?x) - (find jnum- (find (pnum pnum- color) red
                                      (snum snum- city) London)))
(j6 j5 j2)
exec: 0.60 sec     gc: 0.00 sec

; Get S# values for suppliers supplying at least one part supplied by
;       at least one supplier who supplies at least one red part.
* (find (snum- pnum pnum- snum snum- pnum pnum- color) red)
(s3 s4 s2 s5 s1)
exec: 0.36 sec     gc: 0.00 sec

; Get J# values for projects which use only parts which are
;      available from supplier S1.
* ((find jnum- (find qty ?q))
    (find (jnum- pnum) (find pnum- ?r)   (find (pnum- snum) s1)))
nil
exec: 1.21 sec     gc: 0.00 sec
```

4.2. Address Recognition for Mail Sorting.

A research group led by Sargur N. Srihari is studying address-recognition techniques for automated mail sorting (Srihari, Hull et al. 1985). Computer determination of the sort-destination of an arbitrary piece of letter-mail from its visual image is a problem that remains far from completely solved. It involves overcoming several sources of ambiguity at both the spatio-visual and linguistic levels: The location of the destination address has to be determined in the presence of other text and graphics; relevant address lines have to be isolated when there are irrelevant lines of text in the address block; the iconic shapes of characters have to be classified into words of text when numerous types of fonts, sizes, and printing media are present; and the recognized words have to be verified as having the syntax and semantics of an address.

Spatial relationships between objects are essential knowledge sources for vision systems. This source extends naturally to the postal-image understanding problem, because of strong directional expectations. For example, the postage mark is usually above and to the right of the destination address, and the return address is usually to the left of the postage. A semantic network is a natural representation for geometric relations.

An envelope image is segmented into blocks, and a SNePS network is built that represents the geometric relations between blocks and information about the relative and absolute area occupied by each block. A preliminary set of geometric relations are the eight compass points. Relative area occupancy is expressed as the percentage of each block that falls in each of nine equal rectangular subdivisions of the envelope image, and absolute area is given in terms of the number of pixels covered by each block. The program constructs an exhaustive representation of all the geometric relations present in the image. Given the image produced by an initial segmentation procedure, a rough, intuitive output, shown in Figure 6, with some arc labels removed for clarity, was produced.

Future work in this area includes refinement of the data structure to represent more information more efficiently and the addition of inferencing capabilities whose objective is to present the control structure with tentative decisions about the address block based only on the information provided by the initial segmentation.

4.3. NEUREX.

The NEUREX project (Chen 1984, Xiang and Srihari 1985, Xiang et al. 1984, Suchin forthcoming) is a diagnostic expert system for diseases of the central and peripheral nervous systems; it also deals with information about neuroaffectors, neuroreceptors, and body parts. SNePS is used to represent spatial structures and functions propositionally. Entities are represented topologically by means of

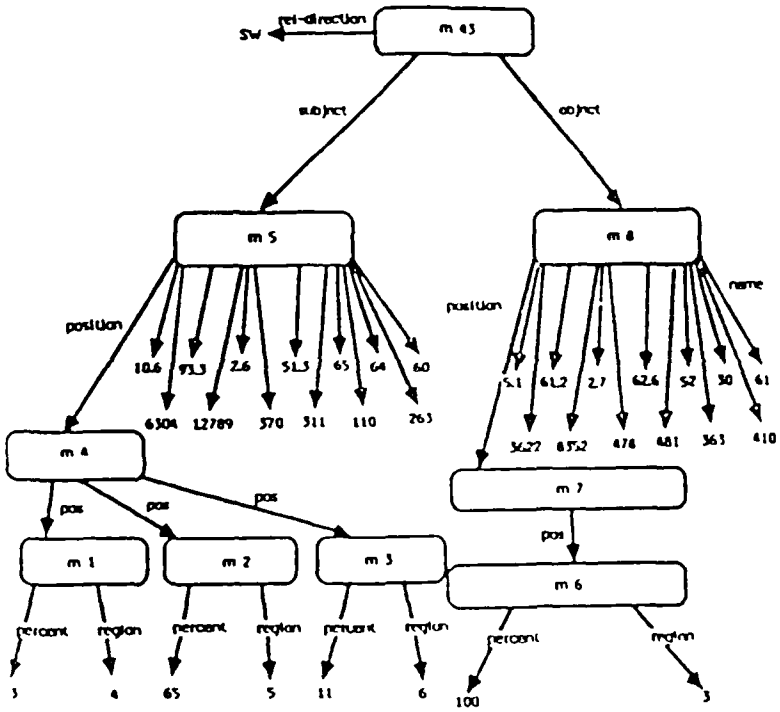


Figure 6. SNePS network representation of initial segmentation of envelope image. (From Srihari, Hull et al. 1985.)

proposition nodes expressing an entity's shape, position, etc., and spatial relations are represented by proposition nodes expressing adjacency, connectivity, direction, etc. This approach integrates structural and functional neuroanatomical information. Moreover, the representation is both propositional and analog: For the peripheral nervous system, there are nodes representing such propositions as that, e.g., a sequence of nerve segments are linked at junctions, and that the whole sequence forms a (peripheral) nerve; the network that is built is itself an analog representation of this nerve (and ultimately, together with its neighbors, of the entire peripheral nervous system). For the central nervous system, there are coordinates in the network representation that can be used to support reasoning by geometrical computation or graphical interfaces.

As one example, the network of Figure 7 can be used by the system to determine which muscles are involved in shoulder-joint flexion, using the SNePS User Language request

(find (ms- cn) (find jt shoulder-joint mv flexion)),

which returns the following list of four nodes:

(deltoid pectoralis_major_clavicular_head coracobrachialis biceps_brachii)

Furthermore, rules, like that shown in Figure 8, can be employed and can even include probabilistic information.

4.4. Representing Graphical Knowledge.

The goal of the Versatile Maintenance Expert System (VMES) project is to develop an expert maintenance system that can reason about digital circuits represented graphically (cf. Shapiro, Srihari et al. 1985). The representation is not pixel-oriented; this is a project in visual knowledge representation integrated with more traditional conceptual and propositional knowledge representation. (The epistemic status of this sort of visual knowledge has not hitherto been specified in the literature.) The graphical form of an object is a LISP function that, when evaluated, draws the object on the screen. Propositional nodes express information about (1) the relative or absolute position of the object and (2) attributes of the object. Visual knowledge can also be distributed among nodes in traditional hierarchies: e.g., the knowledge of how to display a particular hammer may be stored at the level of the class of hammers; the knowledge of how to display a person may be distributed among the nodes for heads, arms, etc.

For example, Figure 9 shows a set of three assertions. Node m233 represents the assertion that the object TRIANGLE-1 is 100 units to the right and 20 units below the object SQUARE-1. The MODALITY arc permits the selection of different modes of display; here, we want to display TRIANGLE-1 in "functional" mode. Node m220 states that every member of the class TRIANGLE displayed in functional mode has the form DTRIANG associated with it. Finally, node m219 asserts that TRIANGLE-1 is a TRIANGLE.

Figure 10 contains four assertions, of which node m246 is the most complex. It links the object GATE-1 to an absolute position at 100/400 and to the class of all AND-gates. Node m244 asserts that GATE-1 is a part of BOARD-1. Node m248 asserts that INP1-GATE1 is a PART-OF GATE-1 and belongs to the class AINP1. The label 'PART' actually stands for "has part". Node m239 links the attribute BAD to GATE-1. Every attribute belongs to an attribute class, and the arc ATTRIBUTE-CLASS points to the class STATE.

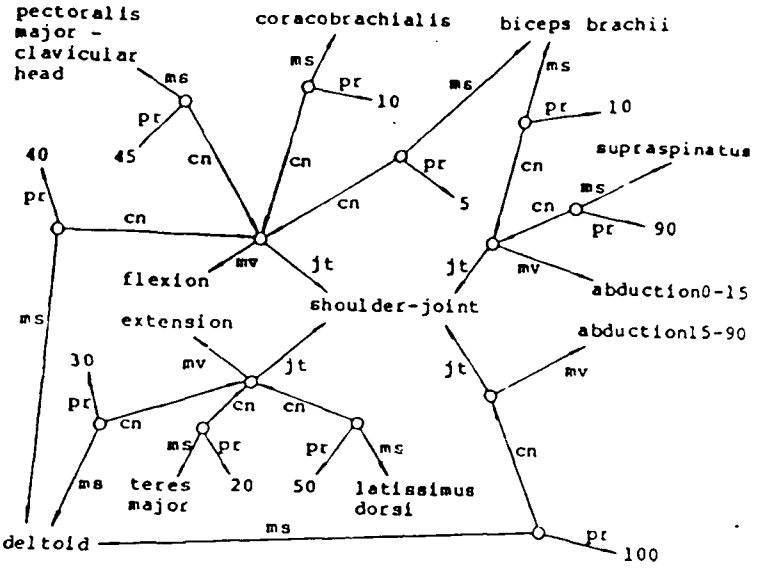
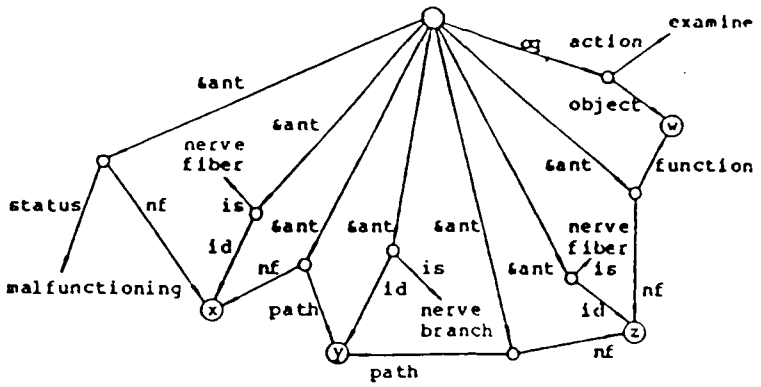


Figure 7. Four of the shoulder-joint movements with muscles involved and their contribution to each relevant movement. (Meaning of arc labels: jt = joint; mv = movement; ms = muscle; cn = contribute; pr = percentage.) (From Xiang and Srihari 1985.)



IF x is a nerve fiber and x is malfunctioning and y is a nerve branch and x goes through y and z is a nerve fiber going through y and z has function w
THEN examine function w.

Figure 8. SNePS network for a NEUREX rule. (From Xiang and Srihari 1985.)

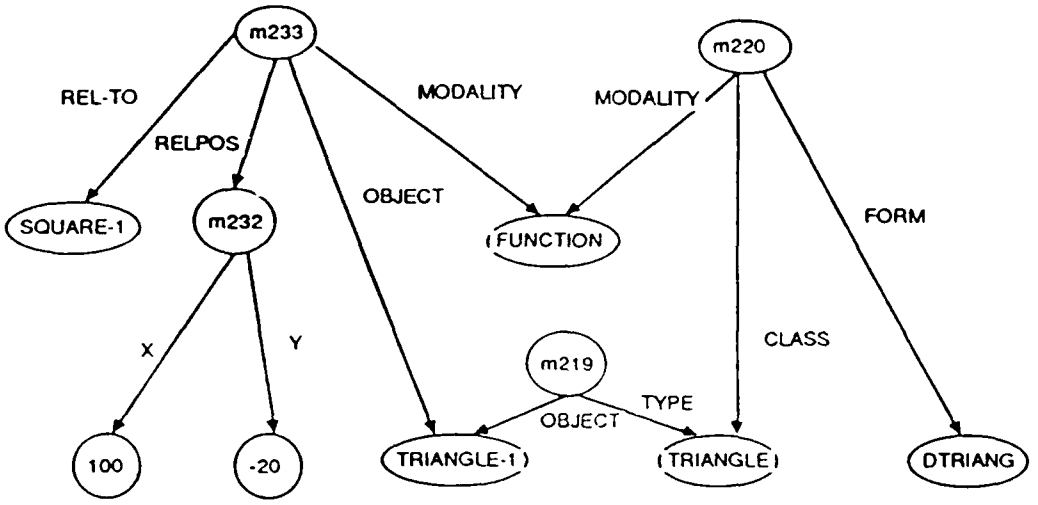


Figure 9. SNePS network in VMES for the form and relative position of TRIANGLE-1.

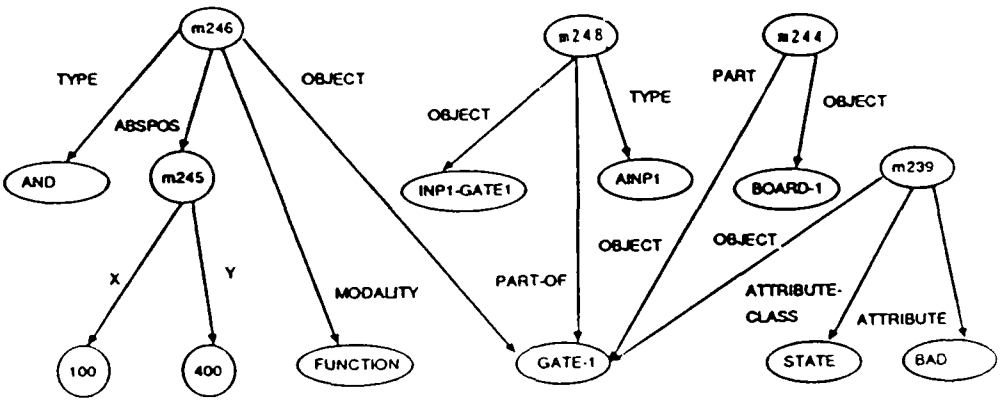


Figure 10. SNePS network in VMES for the location, structure, and state of GATE-1.

4.5. SNeBR: A Belief-Revision Package.

SNePS has been extended by João Martins to handle belief revision—an area of AI research concerned with the issues of revising sets of beliefs when a contradiction is found in a reasoning system. Research topics in belief revision include the study of the representation of beliefs, in particular how to represent the notion of belief dependence; the development of methods for selecting the subset of beliefs responsible for contradictions; and the development of techniques to remove some subset of beliefs from the original set of beliefs. (For an overview of the field, see Martins, forthcoming.)

SNeBR (*SNePS Belief Revision*) is an implementation in SNePS of an abstract belief-revision system called the Multiple Belief Reasoner (MBR), which, in turn, is based on a relevance-logic system called SWM (after Shapiro, Wand, and Martins) (Shapiro and Wand 1976; Martins 1983; Martins and Shapiro 1983, 1984). SWM contains the rules of inference of MBR and defines how contradictions are handled. The only aspect of SWM relevant to this description concerns the objects with which MBR deals, called *supported wffs*. They are of the form

$$A \mid t, o, r$$

where A is a well-formed formula representing a proposition, t is an *origin tag* indicating how A was obtained (e.g., as a hypothesis or as a derived proposition), o is an *origin set* containing *all* and *only* the hypotheses used to derive A, and r is a *restriction set* containing information about contradictions *known* to involve the hypotheses in o. The triple t, o, r is called the *support* of the wff A. The origin tag, origin set, and restriction set of a wff are computed when the wff is derived, and its restriction set may be updated when contradictions are discovered.

MBR uses the concepts of context and belief space. A *context* is any set of hypotheses. A context determines a *belief space*, which is the set of all the hypotheses defining the context together with all propositions derived exclusively from them. The propositions in the belief space defined by a given context are characterized by having an origin set that is contained in the context. At any point, the set of all hypotheses under consideration is called the *current context*, which defines the *current belief space*. The only propositions that are retrievable at a given time are the ones belonging to the current belief space.

A contradiction may be detected either because an assertion is derived that is the negation of an assertion already in the network, or because believed assertions invalidate a rule being used (particularly an AND-OR or a THRESH rule; see (Syn/Sem.11-12)). In the former case, the contradiction is noted when the new, contradictory, assertion is about to be built into the network, since the Uniqueness Principle guarantees that the contradictory assertions will share network structure. In the latter case, the contradiction is noted in the course of applying the rule. In the former case, it may be that the contradictory assertions are in different belief spaces (only the new one being in the current belief space). If so, the restriction sets are updated to reflect the contradictory sets of hypotheses, and nothing else happens. If the contradictory assertions are both in the current belief space (which will be the case when one of them is a rule being used), then, besides updating the restriction sets, the user will be asked to delete at least one of the hypotheses underlying the contradiction from the current context. Management of origin sets according to SWM guarantees that, as long as the current context was originally not known to be contradictory, removal of any one of the hypotheses in the union of the origin sets of the contradictory assertions from the current context will restore the current context to the state of not being known to be inconsistent.

4.6. Knowledge-Based Natural-Language Understanding.

Jeannette Neal (Neal 1985, Neal and Shapiro in press) has developed an AI system that can treat knowledge of its own language as its discourse domain. The system's linguistic knowledge is represented declaratively in its network knowledge base in such a way that it can be used in the dual role of "program" to analyze language input to the system and "data" to be queried or reasoned about. Since language forms (part of) its domain of discourse, the system is also able to learn from the

discourse by being given instruction in the processing and understanding of language. As the system's language knowledge is expanded beyond a primitive kernel language, instructions can be expressed in an increasingly sophisticated subset of the language being taught. Thus, the system's language is used as its own metalanguage.

The kernel language consists of a relatively small collection of predefined terms and rewrite rules for expressing syntax and for expressing the mapping of surface strings to the representation of their interpretations.

The knowledge representations include representations for surface strings and for relations such as: (a) a lexeme being a member of a certain lexical category, (b) bounded string B being in category C and this phrase structure being represented by concept N, (c) a structure or parsed string expressing a certain concept, and (d) one phrase structure being a constituent of another structure.

In order to talk about both the syntax and semantics of language, the network representations distinguish between a word or string and its interpretation. In one experiment, the statements

(1) A WOMAN IS A HUMAN
(2) 'WOMAN IS SINGULAR

were input to the system. The first makes a claim about women; the second makes a claim about the *word* 'woman', as indicated by the quote. Nodes m40 and m50 of Figure 11, respectively, represent the propositions expressed by these statements. The concept or class expressed by 'WOMAN' is represented by node b22; the entity represented by node b22 is a participant in the subset-superset proposition expressed by (1). However, in the representation of (2), the word 'WOMAN' itself is the entity having the property SINGULAR.

Additional statements, such as:

(R) IF THE HEAD-NOUN OF A NOUN-PHRASE X HAS NUMBER Y, THEN X HAS NUMBER Y.

were input to the system to demonstrate the use of a subset of English as its own metalanguage in building up the system's language ability from its primitive predefined language. Figure 12 illustrates the representation of the system's interpretation of rule (R) as well as the representation of certain linguistic relations: Node m87 represents the proposition that some bounded string represented by variable node v4 is in the category HEAD-NOUN, and this phrase structure is represented by variable node v3. Node m88 represents that the phrase structure represented by node v3 is a constituent of v1, which represents a NOUN-PHRASE structure. As soon as any rule such as (R) is parsed and interpreted, it is immediately available for use in subsequent processing. Thus, the system is continuously educable and can use its language as its own metalanguage.

4.7. Temporal Structure of Narrative.

Michael Almeida is using SNePS in the development of a system that will be able to read a simple narrative text and construct a model of its temporal structure (Almeida and Shapiro 1984; Almeida, forthcoming). This project uses an event-based, rather than a proposition-based, approach: i.e., intervals and points of time are associated with events represented as objects in the network rather than with the propositions that describe them. The temporal model itself consists of these intervals and points of time related to one another by such relations as BEFORE, AFTER, DURING/CONTAINS, etc.

The representation of the following short narrative,

> John arrived at the house. The sun was setting. He rang the bell; a minute later, Mary opened the door.

is shown in Figure 13. The ARG-PRED-EVENT case frame asserts that the proposition consisting of the argument pointed to by the ARG-arc and the predicate pointed to by the PRED-arc describes the

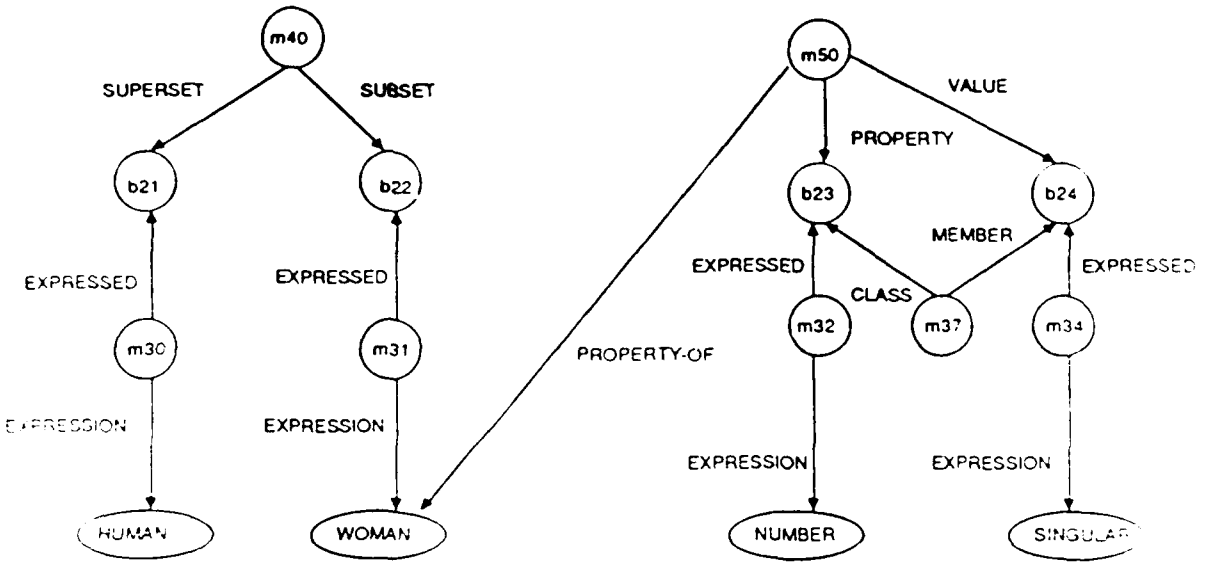


Figure 11. Representation of the interpretation of statements about linguistic and non-linguistic entities.

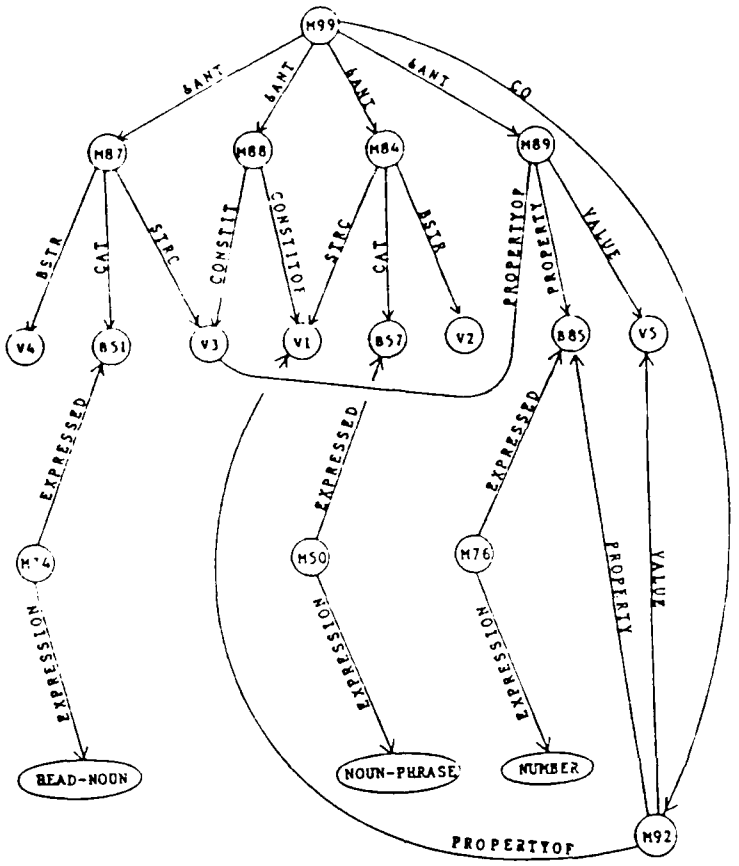


Figure 12. SNePS network for rule (R). (From Neal 1985.)

event pointed to by the EVENT-arc. Notice that the predicates are classified into various types. This information plays an important role in the temporal analysis of a text.

NOW is a reference point that indicates the present moment of the narrative; it is updated as the story progresses through time. NOW is implemented as a variable whose current value is indicated in Figure 13 by a dotted arrow. Subscripts are used in the figure to show the successive values of NOW.

The BEFORE-AFTER-DURATION case frame is used to indicate that the period of time pointed to by the BEFORE-arc temporally precedes the period of time pointed to by the AFTER-arc by the length of time pointed to by the DURATION-arc. These durations are usually not known precisely. The value ε stands for a very short interval; whenever an event occurs in the narrative line, it has the effect of moving NOW an interval of ε beyond it.

The DURING-CONTAINS case frame is used to indicate that the period of time pointed to by the DURING-arc is during (or contained in) the period of time pointed to by the CONTAINS-arc. Notice that the progressive sentence, "The sun was setting", created an event that contains the then-current NOW. If the system knows about such things as sunsets, then it should infer that the event of the sun's setting also contains John's arrival, his ringing of the bell, and probably also Mary's opening of the door.

5. CONCLUSION: SNePS AND CASSIE AS SEMANTIC NETWORKS.

We shall conclude by looking at SNePS from the perspective of Brachman's discussions of structured inheritance networks such as KL-ONE and hierarchies of semantic-network formalisms (Brachman 1977, 1979).

5.1. Criteria for Semantic Networks.

Brachman offers six criteria for semantic-networks:

A semantic network must have a *uniform notation*. SNePS provides some uniform notation with its built-in arc labels for rules, and it provides a uniform procedure for users to choose their own notation.

A semantic network must have an *algorithm for encoding information*. This is provided for by the interfaces to SNePS, e.g., by the parser component of our ATN parser-generator that takes English sentences as input and produces SNePS networks as output.

A semantic network must have an "*assimilation*" *mechanism* for building new information in terms of stored information. SNePS provides for this by the Uniqueness Principle, which enforces node sharing during network building. The assimilation is demonstrated by the generator component of our ATN parser-generator, which takes SNePS nodes as input and produces English output expressing those nodes: Our conversation with CASSIE illustrated this—the node built to represent the new fact, 'Lucy is sweet', is expressed in terms of the already existing node for Lucy (who had previously been described as young) by 'young Lucy is sweet'.

A semantic network should be *neutral* with respect to network formalisms at higher levels in the Brachman hierarchy. SNePS is a semantic network at the "logical" level, whereas KL-ONE is at the "epistemological" level and CASSIE is (perhaps) at the "conceptual" level. SNePS is neutral in the relevant sense; it is not so clear whether CASSIE or KL-ONE are. But neutrality at higher levels may not be so important; a more important issue is the reasons why one formalism should be chosen over another. Several possible criteria that a researcher might consider are: *efficiency* (including the ease of interfacing with other modules; e.g., our ATN parser-generator has been designed for direct interfacing with SNePS), *psychological adequacy* (irrelevant for SNePS, but relevant to some degree for KL-ONE and precisely what CASSIE is being designed for), *ontological adequacy* (irrelevant for CASSIE—see below; claimed for KL-ONE—but see below), *logical adequacy* (guaranteed for SNePS, because of its inference package), and *natural-language adequacy* (a feature of SNePS's interface with the ATN

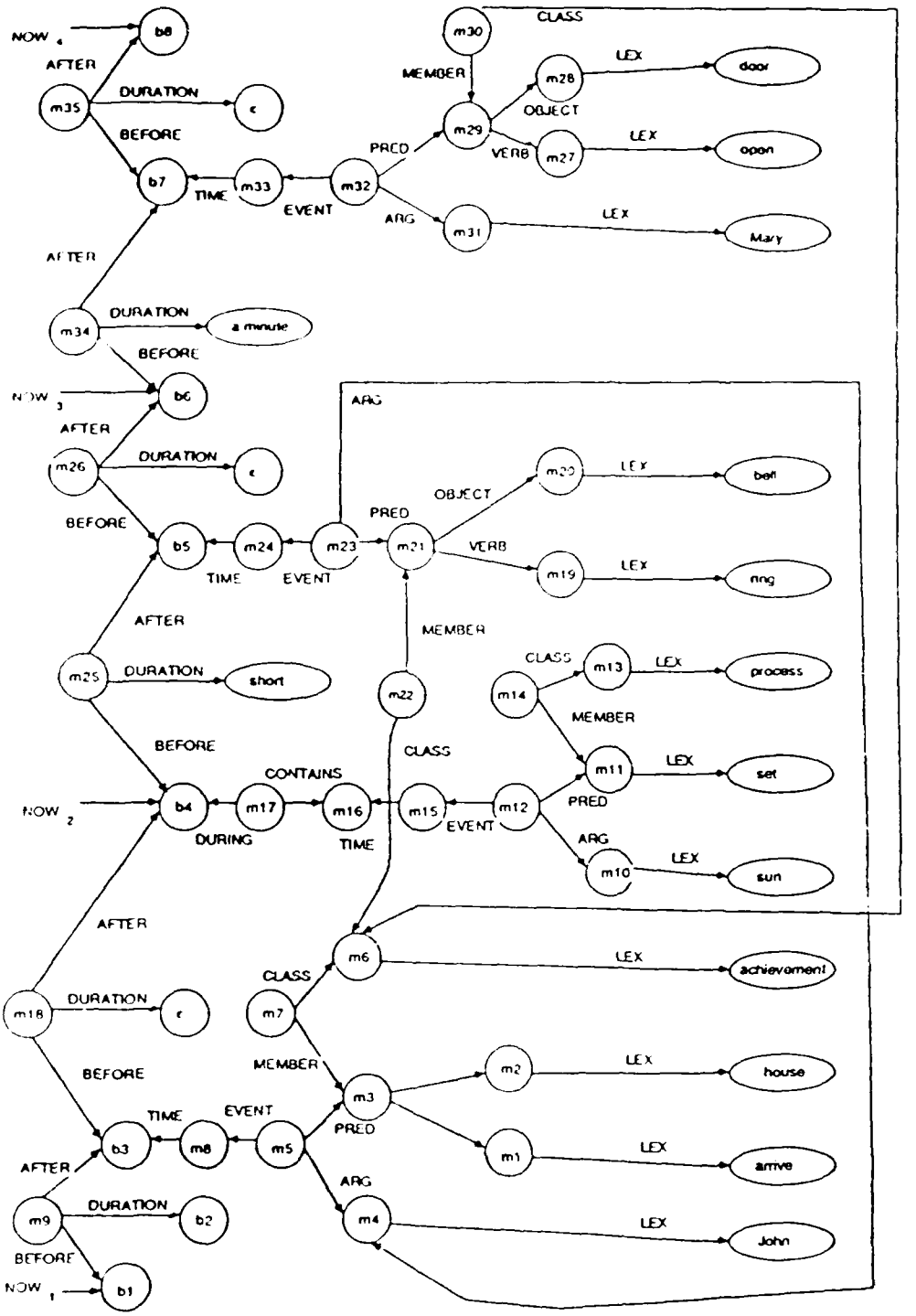


Figure 13. SNePS network for a short narrative.

grammar).

A semantic network should be *adequate* for any higher-level network formalism. SNePS meets this nicely: KL-ONE can be implemented in SNePS (Tranchell 1982).

A semantic network should have a *semantics*. That has been one of the goals of this essay. But it should be observed that there are at least two very different sorts of semantics. In SNePS, nodes have a meaning *within the system* in terms of their links to other nodes; they have a meaning *for users* as provided by the nodes at the heads of LEX arcs. Arcs, on the other hand, only have meaning within the system, provided by node- and path-based inference rules (which can be thought of as procedures that operate on the arcs). In both cases, there is an "internal", system's semantics that is holistic and structural: the meaning of the nodes and arcs are not given in isolation, but in terms of the entire network. This sort of semantics (which actually has more of a syntactic flavor) differs from a semantics that provides links to an external, interpreting system, such as a user or the "world"—i.e., links between the network's way of representing information and the user's way. It is the latter sort of semantics that we provided for CASSIE, above.

5.2. SNePS and CASSIE vs. KL-ONE.

SNePS and CASSIE can be compared directly to KL-ONE. First, KL-ONE is an *inheritance*-network formalism, which represents concepts, instances of concepts, and properties and relations among them. SNePS is a *propositional*-network formalism, which represents propositions and their constituents (individuals, properties, and relations).

But SNePS can handle inheritance, in two ways: We have already seen an example of inheritance by path-based inference in the conversation with CASSIE. In that example, inheritance could also have been accomplished through node-based inference by, e.g., representing 'dogs are animals' as a universally-quantified rule rather than by a SUBCLASS-SUPERCLASS case frame. That is, where an inheritance network might express the claim that dogs are animals by a single arc (say, a subclass-arc) *from a dog-node to an animal-node, SNePS could express it by a proposition* (represented by node m17 in Figure 4b).

One advantage of the propositional mode of representation is that the proposition (m17) expressing the relationship can then become the objective of a proposition representing an agent's belief or it can become the antecedent or consequent of a node-based rule. In some inheritance networks, this could only be done by choosing to represent the entire claim by either the dog-node, the animal-node, the subclass-arc, or (perhaps) the entire structure consisting of the two nodes and the arc. The first two options seem incorrect; the third and fourth either introduce an anomaly into the representation (since arcs can then point either to nodes or to other arcs or to structures), or it reduces to what SNePS does: SNePS, in effect, trades in the single arc for a node with two outgoing arcs. In this way, the arcs of inheritance networks become information-bearing nodes, and the semantic network system becomes a propositional one.

Second, KL-ONE uses "epistemologically primitive links". But why does KL-ONE use the particular set of links that it does, and not some other set; i.e., what is the ontological justification for KL-ONE's links? There have been many philosophical and logical theories of the relations of the One to the Many (part-whole, member-set-superset, instance-concept, individual-species-genus, object-Platonic Form, etc.). KL-ONE's only motivation seems to be as a computationally efficient theory that clarifies the nature of inheritance networks; but it does not pretend to ontological or psychological adequacy. Indeed, it raises almost as many questions as it hopes to answer. For example, in KL-ONE, instances of a general concept seem to consist of *instances* of the attributes of the general concept, each of which instances have *instances* of the values of those attributes. But this begs important philosophical questions about the relations between properties of concepts (or of Forms, or of . . .) and properties of individuals falling under those concepts (or participating in those Forms, or . . . ; some of these issues are discussed in Brachman 1983, but not from a philosophical point of view). Are they the same properties? Are the latter "instances" of the former? Are there such things as concepts (or Forms, or . . .) of properties? And do instance nodes represent individuals? Do they represent individual concepts? (Cf.

Brachman 1977: 148.)

Now, on the one hand, CASSIE's arcs are also taken to be "primitive"; but they are justified by the Meinongian philosophy of mind briefly sketched out above and explored in depth in the references cited. On the other hand, SNePS's arcs, by contrast to both CASSIE's and KL-ONE's, are not restricted to any particular set of primitives: We believe that the interpretation of a particular use of SNePS depends on the *user's* world-view; the user should not be required to conform to *ours*.

And, unlike KL-ONE, the entities in CASSIE's ontology are not to be taken as representing things in the world; CASSIE's ontology is an *epistemological ontology* (cf. Rapaport 1976: 145-49) of the (purely intensional) items that enable a cognitive agent to have beliefs (about the world)—a theory of what there must be in order for a cognitive agent to have beliefs (about what there is).

REFERENCES

(1) Almeida, Michael J., "Reasoning about the Temporal Structure of Narrative Texts," Tech. Report (SUNY Buffalo Dept. of Computer Science, forthcoming).

(2) Almeida, Michael J., and Stuart C. Shapiro, "Reasoning about the Temporal Structure of Narrative Texts," *Proc. Cognitive Science Soc.* 5(1983).

(3) Brachman, Ronald J., "What's in a concept: structural foundations for semantic networks," *Int. J. Man-Machine Studies* 9(1977)127-52.

(4) Brachman, Ronald J., "On the Epistemological Status of Semantic Networks," in N. V. Findler (ed.), *Associative Networks: Representation and Use of Knowledge by Computers* (New York: Academic Press, 1979): 3-50.

(5) Brachman, Ronald J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," *Computer* 16.10(October 1983)30-36.

(6) Brachman, Ronald J.; Richard E. Fikes; and Hector J. Levesque, "KRYPTON: Integrating Terminology and Assertion," *Proc. AAAI* (1983)31-35.

(7) Brachman, Ronald J.; Victoria P. Gilbert; and Hector J. Levesque, "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON," *Proc. IJCAI* 9(1985)532-539.

(8) Carnap, Rudolf, *The Logical Structure of the World* (1928), R. A. George (trans.) (Berkeley: Univ. of California Press, 1967).

(9) Castañeda, Hector-Neri, "Thinking and the Structure of the World" (1972), *Philosophia* 4(1974)3-40; reprinted in 1975 in *Critica* 6(1972)43-86.

(10) Castañeda, Hector-Neri, "Individuals and Non-Identity: A New Look," *American Phil. Qtly.* 12(1975a)131-40.

(11) Castañeda, Hector-Neri, "Identity and Sameness," *Philosophia* 5(1975b)121-50.

(12) Castañeda, Hector-Neri, *Thinking and Doing: The Philosophical Foundations of Institutions* (Dordrecht: D. Reidel, 1975c).

(13) Castañeda, Hector-Neri, "Perception, Belief, and the Structure of Physical Objects and Consciousness," *Synthèse* 35(1977)285-351.

(14) Castañeda, Hector-Neri, "Fiction and Reality: Their Basic Connections," *Poetica* 8(1979)31-62.

(15) Chen, Kweijen, "Representation of Neurological Paths by Semantic Networks: User Interface," Project Report (SUNY Buffalo Dept. of Computer Science, 1984).

(16) Date, C. J., *An Introduction to Database Systems* (Reading, MA: Addison-Wesley, 3rd ed., 1981).

(17) Findlay, J. N., *Meinong's Theory of Objects and Values* (Oxford: Clarendon Press, 2nd ed., 1963).

(18) Fine, Kit, "A Defence of Arbitrary Objects," *Proc. Aristotelian Soc.*, Supp. Vol. 58(1983)55-77.

(19) Lambert, Karel, *Meinong and the Principle of Independence* (Cambridge, Eng.: Cambridge University Press, 1983).

(20) Maida, Anthony S., and Stuart C. Shapiro, "Intensional Concepts in Propositional Semantic Networks," *Cognitive Science* 6(1982)291-330.

(21) Martins, João, "Reasoning in Multiple Belief Spaces," Tech. Report No. 203 (SUNY Buffalo Dept. of Computer Science, 1983).

(22) Martins, João, "Belief Revision," in S. C. Shapiro (ed.), *Encyclopedia of Artificial Intelligence* (New York: John Wiley, forthcoming).

(23) Martins, João, and Stuart C. Shapiro, "Reasoning in Multiple Belief Spaces," *Proc. IJCAI* 8(1983)370-73.

(24) Martins, João, and Stuart C. Shapiro, "A Model for Belief Revision," *Proc. AAAI Non-Monotonic Reasoning Workshop* (1984)241-94.

(25) McCarthy, J., "First Order Theories of Individual Concepts and Propositions," in J. E. Hayes, D. Michie, and L. Mikulich (eds.), *Machine Intelligence* 9(Chichester, Eng.: Ellis Horwood, 1979): 129-47.

(26) McKay, Donald P., and João Martins, "SNePSLOG User's Manual," SNeRG Tech. Note No. 4 (SUNY Buffalo Dept. of Computer Science, 1981).

(27) McKay, Donald P., and Stuart C. Shapiro, "MULTI—A LISP-Based Multiprocessing System," *Proc. 1980 LISP Conf.* (Stanford: Stanford Univ., 1980): 29-37.

(28) McKay, Donald P., and Stuart C. Shapiro, "Using Active Connection Graphs for Reasoning with Recursive Rules," *Proc. IJCAI* 7(1981)368-74.

(29) Meinong, Alexius, "Über Gegenstandstheorie" (1904), in R. Haller (ed.), *Alexius Meinong Gesamtausgabe*, Vol. II (Graz, Austria: Akademische Druck- u. Verlagsanstalt, 1971): 481-535. English translation ("The Theory of Objects") by I. Levi et al., in R. M. Chisholm (ed.), *Realism and the Background of Phenomenology* (New York: Free Press, 1960): 76-117.

(30) Meinong, Alexius, *On Assumptions* (1910), J. Heanue (ed. and trans.) (Berkeley: Univ. of California Press, 1983).

(31) Morgado, Ernesto J. M., "Semantic Networks as Abstract Data Types," Tech. Report (SUNY Buffalo Dept. of Computer Science, forthcoming).

(32) Neal, Jeannette G., *A Knowledge Based Approach to Natural Language Understanding*, Tech. Report No. 85-06 (SUNY Buffalo Dept. of Computer Science, 1985).

(33) Neal, Jeannette G., and Stuart C. Shapiro, "Knowledge Based Parsing," in L. Bolc (ed.), *Natural Language Parsing Systems* (Berlin: Springer-Verlag, in press).

(34) Parsons, Terence, *Nonexistent Objects* (New Haven: Yale Univ. Press, 1980).

(35) Rapaport, William J., *Intentionality and the Structure of Existence*. Ph.D. diss. (Indiana Univ. Dept. of Philosophy, 1976).

(36) Rapaport, William J., "Meinongian Theories and a Russellian Paradox," *Noûs* 12(1978)153-80; errata, *Noûs* 13(1979)125.

(37) Rapaport, William J., "How to Make the World Fit Our Language: An Essay in Meinongian Semantics," *Grazer Phil. Studien* 14(1981)1-21.

(38) Rapaport, William J., "Meinong, Defective Objects, and (Psycho-)Logical Paradox," *Grazer Phil. Studien* 18(1982)17-39.

(39) Rapaport, William J., Critical Notice of Routley 1979, *Phil. and Phenomenological Research* 44(1984a)539-52.

(40) Rapaport, William J., "Belief Representation and Quasi-Indicators," Tech. Report No. 215 (SUNY Buffalo Dept. of Computer Science, 1984b).

(41) Rapaport, William J., Review of Lambert 1983, Tech. Report 217 (SUNY Buffalo Dept. of Computer Science, 1984c); forthcoming in *J. Symbolic Logic*.

(42) Rapaport, William J., "Meinongian Semantics for Propositional Semantic Networks," *Proc. ACL* 23(1985a)43-48.

(43) Rapaport, William J., "To Be and Not to Be," *Noûs* 19(1985b).

(44) Rapaport, William J., "Meinongian Semantics and Artificial Intelligence," in P. Simons (ed.), *Essays on Meinong* (Munich: Philosophia Verlag, forthcoming).

(45) Rapaport, William J., and Stuart C. Shapiro, "Quasi-Indexical Reference in Propositional Semantic Networks," *Proc. 10th Int. Conf. Computational Linguistics* (COLING-84) (Morristown, NJ: Assoc. Computational Linguistics, 1984): 65-70.

(46) Routley, Richard, *Exploring Meinong's Jungle and Beyond* (Canberra: Australian National University, Research School of Social Sciences, Dept. of Philosophy, 1979).

(47) Shapiro, Stuart C. *The MIND System: A Data Structure for Semantic Information Processing*, Report No. R-837-PR (Santa Monica, CA: The Rand Corporation, 1971a). Also AD No. 733 560, Defense Documentation Center, Alexandria, VA.

(48) Shapiro, Stuart C., "A Net Structure for Semantic Information Storage, Deduction and Retrieval," *Proc. IJCAI* 2(1971b)512-523.

(49) Shapiro, Stuart C., "Representing and Locating Deduction Rules in a Semantic Network," *Proc. Workshop on Pattern-Directed Inference Systems, SIGART Newsletter*, No. 63 (1977): 14-18.

(50) Shapiro, Stuart C., "Path-Based and Node-Based Inference in Semantic Networks," in D. Waltz (ed.), *Tinlap-2: Theoretical Issues in Natural Language Processing* (New York: ACM, 1978)219-225.

(51) Shapiro, Stuart C., "The SNePS Semantic Network Processing System," in N. V. Findler (ed.), *Associative Networks: The Representation and Use of Knowledge by Computers* (New York: Academic Press, 1979a): 179-203.

(52) Shapiro, Stuart C., "Numerical Quantifiers and Their Use in Reasoning with Negative Information," *Proc. IJCAI* 6(1979b)791-96.

(53) Shapiro, Stuart C., "Generalized Augmented Transition Network Grammars for Generation from Semantic Networks," *American J. Computational Linguistics* 8.1 (January-March 1982)12-25.

(54) Shapiro, Stuart C.; João Martins; and Donald P. McKay, "Bi-Directional Inference," *Proc. Cognitive Science Soc.* 4(1982)90-93.

(55) Shapiro, Stuart C., and Donald P. McKay, "Inference with Recursive Rules," *Proc. AAAI* 1(1980)151-53.

(56) Shapiro, Stuart C.; Donald P. McKay; João Martins; and Ernesto Morgado, "SNePSLOG: A 'Higher Order' Logic Programming Language," presented at the 1981 Workshop on Logic Programming for Intelligent Systems, Long Beach, CA; SNeRG Tech. Note No. 8 (SUNY Buffalo Dept. of Computer Science, 1981).

(57) Shapiro, Stuart C.; Sargur N. Srihari; Ming-Ruey Taie; and James Geller, "Development of an Intelligent Maintenance Assistant," *SIGART Newsletter*, No. 92, (April 1985): 48-49.

(58) Shapiro, Stuart C., and Mitchell Wand, "The Relevance of Relevance," Tech. Report No. 46 (Indiana Univ. Dept. of Computer Science, 1976).

(59) Shapiro, Stuart C., and G. H. Woodmansee, "A Net Structure Based Relational Question Answerer: Description and Examples," *Proc. IJCAI* 1(1969)325-346.

(60) Shapiro, Stuart C.; G. H. Woodmansee; and M. W. Kreuger, "A Semantic Associational Memory Net That Learns and Answers Questions (SAMENLAQ)," Tech. Report No. 8 (Univ. of Wisconsin Computer Sciences Dept., 1968).

(61) Srihari, Rohini K., "Combining Path-Based and Node-Based Inference in SNePS," Tech. Report No. 183 (SUNY Buffalo Dept. of Computer Science, 1981).

(62) Srihari, Sargur N.; Jonathan J. Hull; Paul W. Palumbo; Debashish Niyogi; and Ching-Huei Wang, "Address Recognition Techniques in Mail Sorting: Research Directions," Tech. Report No. 85-09 (SUNY Buffalo Dept. of Computer Science, 1985).

(63) Suchin, Jennifer, "A Semantic Network Representation of the Peripheral Nervous System," Project Report (SUNY Buffalo Dept. of Computer Science, 1985).

(64) Tomberlin, James E. (ed.), *Agent, Language, and the Structure of the World* (Indianapolis: Hackett, 1984).

(65) Tranchell, Lynn M., "A SNePS Implementation of KL-ONE," Tech. Report No. 198 (SUNY Buffalo Dept. of Computer Science, 1982).

(66) Woods, William A., "What's in a Link: Foundations for Semantic Networks," in D. G. Bobrow and A. M. Collins (eds.), *Representation and Understanding: Studies in Cognitive Science* (New York: Academic Press, 1975): 35-82.

(67) Xiang, Zhigang, and Sargur N. Srihari, "Spatial Structure and Function Representation in Diagnostic Expert Systems," *Proc. Int. Workshop on Expert Systems and Their Applications* 5(1985)191-206.

(68) Xiang, Zhigang; Sargur N. Srihari; Stuart C. Shapiro; and Jerry G. Chutkow, "Analogical and Propositional Representations of Structure in Neurological Diagnosis," *Proc. Conference on Artificial Intelligence Applications* 1(1984)127-32.

(69) Zalta, Edward, *Abstract Objects* (Dordrecht: D. Reidel, 1983).

BELIEF REVISION IN SNePS[†]

by

João P. Martins[*] and Stuart C. Shapiro[**]

[*]Departamento de Engenharia Mecanica
Instituto Superior Tecnico
Av. Rovisco Pais
1000 Lisboa, Portugal

[**]Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260, U.S.A

ABSTRACT

SNePS is a powerfull knowledge representation system which allows multiple beliefs (beliefs from multiple agents, contradictory beliefs, hypothetical beliefs) to be simultaneously represented, and performs both forward and backward reasoning within sets of these beliefs. SNeBR, described in this paper, is a belief revision package available in SNePS. SNeBR relies on a logic developed to support belief revision systems, the SWM system, and its implementation relies on the manipulation of assumptions, rather than justifications, as is common in other belief revision systems. The first aspect guarantees, among other things, that every proposition in SNeBR is associated with those (and only those) hypotheses from which it was derived; The second aspect enables it to effectively switch reasoning contexts and to avoid having to "mark" every proposition which should not be considered by the knowledge base retrieval operation.

† This work was partially supported by the National Science Foundation under Grant MCS8004314 and by the Instituto Nacional de Invertigação Cientifica (Portugal), under Grant no.20536; Preparation of this paper was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008.

INTRODUCTION

SNePS (Semantic Network Processing System) [Shapiro 79a] is a powerfull knowledge representation system which allows multiple beliefs (beliefs from multiple agents, contradictory beliefs, hypothetical beliefs) to be simultaneously represented, and performs both forward and backward reasoning within sets of these beliefs. In this paper, we discuss SNeBR (**SNePS Belief Revision**), a belief revision system available in SNePS.

Belief revision systems are AI programs that can detect and recover from contradictions. Belief revision systems have been implemented by several researchers (e.g., [Doyle 79; Martins 83; McAllester 80; Steels 80]). It has been argued that a belief revision system relying on the manipulation of assumptions[1] has multiple advantages over one relying in the manipulation of justifications[2] [Martins 83], [Martins and Shapiro 83], [deKleer 84]. A difficulty associated with assumption-based belief revision systems is that it must be possible to compute *exactly* which assumptions underlie a given proposition. SNeBR relies on the manipulation of assumptions, and is based on a logic, the SWM system, which guarantees that every proposition is associated with exactly every hypothesis used in its derivation[3]

In this paper we briefly introduce SNeBR and its underlying system, SWM, and show an example obtained using SNeBR. SNeBR is fully implemented in Franz Lisp, running on VAX-11 Systems.

THE SWM SYSTEM -- THEORETICAL FOUNDATIONS

The SWM[4] system [Martins 83] is the logical system that provides the theoretical foundations for SNeBR. It is loosely based on the relevance logic systems of [Anderson and Belnap 75] and [Shapiro and Wand 76]. Distinguishing features of SWM include recording dependencies of

[1]These systems associate each proposition with the hypotheses (non-derived propositions) that underlie it.

[2]These systems associate each proposition with the propositions that *directly* originated it.

[3]SWM guarantees much more than just this, see [Martins 83].

[4]After Shapiro, Wand and Martins.

wffs, not allowing irrelevancies to be introduced, and providing for dealing with contradictions.

SWM deals with objects called *supported wffs*. Supported wffs are of the form $F \mid \tau,\alpha,\rho$, in which F is a wff (well formed formula), τ (the origin tag) is an element of the set {hyp, der, ext}, α (the origin set) is a set of hypotheses, and ρ (the restriction set) is a set of sets of hypotheses. The *origin set* contains all the hypotheses which were *actually* used in the derivation of F. The *origin tag* tells whether F is an hypotheses (τ=hyp), a normally derived wff (τ=der) or a wff with an extended origin set (τ=ext)[5]. The *restriction set* contains sets of hypotheses, each of which when unioned with the hypotheses in the origin set forms a set which is *known to be inconsistent*.[6]

The rules of inference of the SWM system (see, for example [Martins and Shapiro 84]), guarantee that:

1. The origin set of a supported wff contains *every* hypothesis that was used in its derivation.
2. The origin set of a supported wff contains *only* the hypotheses that were used in its derivation.
3. The restriction set of a supported wff records *every* set known to be inconsistent with the wff's origin set.
4. The application of rules of inference is blocked if the resulting wff would have an origin set known to be inconsistent.

CONTEXTS AND BELIEF SPACES

[5] This latter case will not be discussed in this paper and can be found in [Martins 83] and [Martins and Shapiro 84].

[6] An inconsistent set is a set from which a contradiction may be derived. A set is known to be inconsistent if it is an inconsistent set and a contradiction *is* derived from it.

SNeBR relies on the notions of context and belief space. A *context* is a set of hypotheses. A context determines a *Belief Space* (BS) which is the set of all the hypotheses defining the context and all the propositions which were derived from them. Within SWM, the propositions in a given BS are characterized by having an origin set which is contained in the context.

Any query to the network is associated with a context. When answering the query SNeBR only considers the propositions in the network which belong to the BS defined by that context.

NON-STANDARD CONNECTIVES

SNePS has a a powerful set of non-standard connectives [Shapiro 79a, 79b; Martins and Shapiro, forthcoming]. The disadvantage in using the standard connectives ($\wedge$, $\vee$, $\rightarrow$, $\neg$) relates to the fact that all the connectives, except negation, are binary and therefore expressing sentences about sets of propositions becomes cumbersome. For example, suppose that given three propositions, say A, B and C, we wanted to express the fact that exactly one of them is true. Using the standard connectives this would be done as $(A\wedge\neg B\wedge\neg C) \vee (\neg A\wedge B\wedge\neg C) \vee (\neg A\wedge\neg B\wedge C)$, which is lengthy and difficult to read. Sentences involving more than three propositions are even more complicated and this type of sentence often occurs in some of the intended applications.[7] The SNePS connectives generalize the standard logical connectives to take sets of propositions. In this paper we discuss two of them: and-or and thresh.

And-or is a connective which generalizes $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\oplus$ (exclusive or), $\mid$ (nand) and $\downarrow$ (nor). *And-or*, written ${}_n i^j$, takes as arguments a set of n propositions. The proposition represented by the wff ${}_n i^j(P_1, \ldots, P_n)$ asserts that there is a relevant connection between $P_1, \ldots, P_n$ such that at least i and at most j of them must simultaneously be true. In other words, if $n-i$ arguments of ${}_n i^j$ are false, then the remaining i have to be true and if j arguments of ${}_n i^j$ are true then the remaining $n-j$ have to be false. That and-or is some of

[7] For example, exactly five out of ten propositions are true. Refer to the section on selecting between alternatives.

the generalizations that we claim can be seen by the following:

$$_{1}0(A) = \neg A$$

$$_{2}2(A,B) = A \wedge B$$

$$_{2}1 = A \vee B$$

$$_{2}1(A,B) = AB$$

$$_{2}0(A,B) = A \downarrow B$$

$$_{2}0(A,B) = A \mid B$$

Thresh generalizes equivalence to take a set of arguments. *Thresh*, written $_{ni}$ takes as arguments a set of n propositions. The proposition represented by the wff $_{ni}(P_1, \ldots, P_n)$ asserts that there is a relevant connection between $P_1, \ldots, P_n$ such that either fewer than i of them are true or they all are true. In other words, if at least i of the arguments of $_{ni}$ are true then all the remaining arguments have to be true and if $i-1$ arguments of $_{ni}$ are true and at least one is false, then the remaining arguments have to be false. Equivalence is expressed by $_{n1}(A_1, \ldots, A_n)$.

THE INFERENCE SYSTEM

The SNePS inference system has the following characteristics: it allows both backward and forward inference to be performed; every deduction rule[5] in the network may be used in either backward or forward inference or both; when a deduction rule is used it is activated and remains that way until explicit de-activated by the user; the activated rules are assembled into a set of processes, called an *active connection graph* (acg) [McKay and Shapiro 80], which carry out the inferences; the acg also stores all the results generated by the activated rules; if during some deduction, the inference system needs some of the rules activated during a

[5]We use the term *deduction rule* to refer to any proposition which has either a connective or a quantifier (or both). A deduction rule is a statement in the object language, and can be considered a recipe, plan or heuristic for deriving new information from old information.

previous deduction it uses their results directly instead of re-deriving them [Shapiro, Martins and McKay 82].

There are two main concepts involved in the implementation of the inference package: pattern-matching and the use of procedural (or active) versions of deduction rules.

The *pattern-matching process* is given a piece of the network (either to be deduced in backward inference or added in forward inference) and a context, and locates relevant deduction rules in the BS defined by the context. Such deduction rules are then "compiled" into a set of processes which are given to a multi-processing system for execution. The *multi-processing system* used by SNePS, called MULTI [McKay and Shapiro 80][*], is a LISP based system mainly consisting of a simple evaluator, a scheduler and system primitives. The evaluator continuously executes processes from a process queue until the queue becomes empty; the scheduler inserts processes into the process queue; system primitives include functions for creating processes, scheduling processes and for manipulating local variables or registers. Every process has a name which defines the action the process will perform and also has a continuation link naming the process that is to be scheduled for activation after it has completed its job. There are MULTI processes to perform the following tasks: To match a given structure against the network in the BS defined by some context; To receive answers and to remember all the answers received. To perform the elimination of the main connective of a deduction rule; etc.

For a detailed description of the processes and the form of the acg built during inference refer to [McKay and Shapiro 80], [Martins 83] and [Shapiro Martins and McKay 82].

AN ANNOTATED EXAMPLE -- SELECTING BETWEEN ALTERNATIVES

We present an example of person-machine interaction by showing how SNeBR obtains the solution to the puzzle, named "The Woman Freeman Will Marry", from [Summers 72]. A

[*]The multi processing approach was influenced both by Kaplan's producer-consumer model [Kaplan 73] and by Wand's frame model of computation [Wand 74].

characteristic of this puzzle is that there is no straightforward path from the propositions in the puzzle's statement to the puzzle's solution. In solving this puzzle one has to raise hypotheses, reason from them and if a contradiction is detected replace some of those hypotheses and resume the reasoning. The statement of the puzzle is as follows:

> Freeman knows five women: Ada, Bea, Cyd, Deb and Eve. The women are in two age brackets: three women are under 30 and two women are over 30. Two women are teachers and the other three women are secretaries. Ada and Cyd are in the same age bracket. Deb and Eve are in different age brackets. Bea and Eve have the same occupation. Cyd and Deb have different occupations. Of the five women, Freeman will marry the teacher over 30. Who will Freeman marry?

Figure 1 shows the representation of every proposition in the puzzle's statement.[10] The wffs are described in a language called SNePSLOG [McKay and Martins 81] which is a logic programming interface to SNePS. Assertions and rules written in SNePSLOG are stored as structures in the SNePS network; SNePSLOG queries are translated into top-down deduction requests to the inference system; output from the inference is translated into SNePSLOG formulas for printing to the user.

In Figure 1 we represent the following propositions: There are five women, Ada, Bea, Cyd, Deb and Eve (wff1, wff2, wff3, wff4, wff5). Three women are under 30 (wff12)[11] and two women are over 30 (wff18). Every woman is either under 30 or over 30 (wff27).[12] Two women are teachers (wff33) and the other three women are secretaries (wff39). The *the* in the previous sentence conveys the information that no woman is both a teacher and a secretary, represented by wff48. Ada and Cyd are in the same age bracket (wff53). Deb and Eve are in

---

[10]The numbers associated with the wffs relate to the number of the node which represents the wff in the network.

[11]With this proposition we can see the advantage of the SNePS connectives. With the standard connectives this proposition would be expressed in the following way: (~age(Ada,u=30) ∧ ~age(Bea,u=30) ∧ age(Cyd,u=30) ∧ age(Deb,u=30) ∧ age(Eve,u=30))∨ (~age(Ada,u=30) ∧ age(Bea,u=30) ∧ ~age(Cyd,u=30) ∧ age(Deb,u=30) ∧ age(Eve,u=30))∨ (~age(Ada,u=30) ∧ age(Bea,u=30) ∧ age(Cyd,u=30) ∧ ~age(Deb,u=30) ∧ age(Eve,u=30))∨ (~age(Ada,u=30) ∧ age(Bea,u=30) ∧ age(Cyd,u=30) ∧ age(Deb,u=30) ∧ ~age(Eve,u=30))∨ (age(Ada,u=30) ∧ ~age(Bea,u=30) ∧ ~age(Cyd,u=30) ∧ age(Deb,u=30) ∧ age(Eve,u=30))∨ (age(Ada,u=30) ∧ ~age(Bea,u=30) ∧ age(Cyd,u=30) ∧ ~age(Deb,u=30) ∧ age(Eve,u=30))∨ (age(Ada,u=30) ∧ ~age(Bea,u=30) ∧ ~age(Cyd,u=30) ∧ age(Deb,u=30) ∧ ~age(Eve,u=30))∨ (age(Ada,u=30) ∧ age(Bea,u=30) ∧ ~age(Cyd,u=30) ∧ age(Deb,u=30) ∧ ~age(Eve,u=30))∨ (age(Ada,u=30) ∧ age(Bea,u=30) ∧ age(Cyd,u=30) ∧ ~age(Deb,u=30) ∧ ~age(Eve,u=30))

[12]This information is implicitly contained in the statement of the puzzle.

wff1 : $Woman(Ada)$

wff2 : $Woman(Bea)$

wff3 : $Woman(Cyd)$

wff4 : $Woman(Deb)$

wff5 : $Woman(Eve)$

wff12 : ${}_{55}^{3}(age(Ada,\mu 30),age(Bea,\mu 30),age(Cyd,\mu 30),age(Deb,\mu 30),age(Eve,\mu 30))$

wff18 : ${}_{55}^{3}(age(Ada,o30),age(Bea,o30),age(Cyd,o30),age(Deb,o30),age(Eve,o30))$

wff27 : $\forall x Woman(x) \rightarrow {}_{21}^{1}(age(x,\mu 30),age(x,o30))$

wff33 : ${}_{55}^{3}(worker(Ada,teacher),worker(Bea,teacher),worker(Cyd,teacher),$
$worker(Deb,teacher),worker(Eve,teacher))$

wff39 : ${}_{55}^{3}(worker(Eve,secretary),worker(Deb,secretary),worker(Cyd,secretary),$
$worker(Bea,secretary),worker(Ada,secretary))$

wff48 : $\forall(x)Woman(x) \rightarrow {}_{21}^{1}(worker(x,secretary),worker(x,teacher))$

wff53 : $\forall(x)_{21}^{1}(age(Ada,x),age(Cyd,x))$

wff58 : $\forall(x)_{21}^{1}(age(Deb,x),age(Eve,x))$

wff63 : $\forall(x)_{21}^{1}(worker(Bea,x),worker(Eve,x))$

wff68 : $\forall(x)_{21}^{1}(worker(Cyd,x),worker(Deb,x))$

wff79 : ${}_{51}^{1}({}_{22}^{2}(age(Ada,o30),worker(Ada,teacher)),$

${}_{22}^{2}(age(Bea,o30),worker(Bea,teacher)),$

${}_{22}^{2}(age(Cyd,o30),worker(Cyd,teacher)),$

${}_{22}^{2}(age(Deb,o30),worker(Deb,teacher)),$

${}_{22}^{2}(age(Eve,o30),worker(Eve,teacher)))$

wff88 : $\forall(x)_{21}^{1}(marry(Freeman,x),{}_{22}^{2}(age(x,o30),worker(x,teacher)))$

Figure 1
Propositions in the network

different age brackets (wff58). Bea and Eve have the same occupation (wff63). Cyd and Deb

have different occupations (wff68). Exactly one woman over 30 is a teacher (wff79). Freeman will marry the teacher over 30 (wff88).

To solve the puzzle we raise hypotheses about the ages and professions of the women and ask SNeBR to deduce who Freeman will marry under those assumptions. If the hypotheses raised are consistent with the puzzle's statement the desired answer will be returned, otherwise a contradiction will be detected and SNeBR will guide us in discarding hypotheses.

wff6 : $_{55}^{5}(wff5,wff4,wff3,wff2,wff1)$ | hyp,{wff6} ,{}

wff89 : $_{12}{}_{15}^{12}(wff88,wff79,wff68,wff63,wff58,wff53,wff48,wff39,wff33,$
$wff27,wff18,wff12)$ | hyp,{wff89} ,{}

wff13 : $age(Ada,o30)$ | hyp,{wff13} ,{}

wff15 : $age(Cyd,o30)$ | hyp,{wff15} ,{}

wff28 : $worker(Ada,teacher)$ | hyp,{wff28} ,{}

wff31 : $worker(Deb,teacher)$ | hyp,{wff31} ,{}

Figure 2
Hypotheses raised

Using the propositions described in Figure 1, we built into the network the hypotheses represented in Figure 2. The hypothesis represented by wff6 states that there are five women and names those women, and the hypothesis represented by wff89 asserts all the specific information pertaining these women and their relationship with Freeman. The hypotheses represented by wff13, wff15, wff28, and wff31 define the ages and professions of the women.[1] Suppose that we ask who Freeman will marry under the BS defined by the context

[1] Notice that specifying the ages of the two women over 30 completely determines the ages of the five women, and that specifying the names of the two women who are teachers completely determines the profession of the five women.

{wff6, wff13, wff15, wff28, wff31, wff 89}. In this BS there is no assertion about who Freeman will marry but wff88 may enable its deduction. SNeBR sets up two sub-goals, finding who is over 30 and finding who is a teacher (Figure 3).

I wonder if *marry(Freeman,who)*
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

let me try to use the rule $\forall(x)_{21}(marry(Freeman,x),_{22}^{2}(age(x,o30),worker(x,teacher)))$

I wonder if *age(x,o30)*
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

I know *age(Cyd,o30)*

I know *age(Ada,o30)*

I wonder if *worker(x,teacher)*
holds within the BS defined by the context (wff31 wff28 wff15 wff13 wff89 wff6)

I know *worker(Deb,teacher)*

I know *worker(Ada,teacher)*

since *worker(Ada,teacher)* and *age(Ada,o30)* I infer *marry(Freeman,Ada)*

Figure 3
Ada and Cyd are over 30; Ada and Deb are teachers
Freeman will marry Ada

Figure 3, shows SNeBR's deduction that Freeman will marry Ada. The inference does not stop here, however, since there are several processes still waiting for answers and SNeBR reports inferences as shown in Figure 4.

since $age(Ada,o30)$ and $age(Cyd,o30)$

I infer $_{10}^{0}(age(Bea,o30))$ $_{10}^{0}(age(Deb,o30))$ $_{10}^{0}(age(Eve,o30))$

since not $age(Eve,o30)$ I infer $age(Deb,o30)$

Figure 4
Bea, Deb and Eve are not over 30
Deb is over 30

After the deduction of the information shown in Figure 4, a contradiction is detected (Figure 5). A contradiction will be detected by SNeBR when one of the following conditions occurs: 1) Nodes representing contradictory wffs are built into the BS under consideration;[14] 2) Information gathered by a connective elimination process shows that a rule is invalidated by the data in the BS.

In our example this latter case occurs: there exists one process to deduce information using the rule $_{2}^{3}(age(Ada,o30),age(Bea,o30),age(Cyd,o30),age(Deb,o30),age(Eve,o30))$ which gathers that there are three women who are over 30 (Ada, Cyd and Deb).

[14] If nodes representing contradictory propositions are built but none of them does not belong to the BS under consideration, SNeBR reports that there is an inconsistent BS (which is not being considered) and proceeds. Refer to (Martins & Shapiro, 83).

WARNING!
Contradiction detected in the following and-or
$_{52}^{2}$(*age*(*Ada*,*o*30),*age*(*Bea*,*o*30),*age*(*Cyd*,*o*30),*age*(*Deb*,*o*30),*age*(*Eve*,*o*30))
More true arguments than max.
Arguments in wrong number *age*(*Ada*,*o*30) *age*(*Cyd*,*o*30) *age*(*Deb*,*o*30)
You have the following options:
1. Continue anyway, knowing that a contradiction is derivable;
2. Re-start the exact same request in a different context which is not inconsistent;
3. Drop the request altogether.
Do you want to continue anyway?
=><= n
Do you want to re-start the request in a new context?
=><= yes

Figure 5
A contradiction is detected

Upon detecting the contradiction SNeBR gives the options of continuing the reasoning within the inconsistent BS,[15] modifying the current context in order to obtain a consistent BS or giving up the request. In our example, we decided to restore consistency causing the interaction shown in Figures 6 and 7.[16]

[15] In SNeBR this is not dangerous since it is based on relevance logic in which the paradoxes of implication (e.g. from a contradiction anything can be derived) do not arise.

[16] Note that the restriction set of this extended wff has the set {wff13, wff15}, meaning that {wff13, wff25, wff59, wff15} is a set known to be inconsistent.

```
In order to make the context consistent you must delete some hypotheses from the set
(wff13 wff15 wff89)
You are now entering a package that will enable you to delete some hypotheses from this set.

Do you want to take a look at wff13 ?
=><= n
There are 5 propositions depending on wff13 : (wff97 wff16 wff93 wff91 wff90) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= a
  10(marry(Freeman,Eve)) | ext,{wff13,wff28,wff89} ,{{wff15}}
                .
                .

What do you want to do with wff13 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= d

Do you want to take a look at wff15 ?
=><= y
   age(Cyd,o30) | hyp,{wff15} ,{{wff13,wff89}}
There are 2 propositions depending on wff15 : (wff16 wff91) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= n
What do you want to do with wff15 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= d

Do you want to take a look at wff89 ?
=><= n
There are 8 propositions depending on wff89 :
(wff97 wff95 wff16 wff94 wff93 wff92 wff91 wff90) .
Do you want to look at [a]ll of them, [s]ome of them, or [n]one?
=><= n
What do you want to do with wff89 ?
[d]iscard from the context, [k]eep in the context, [u]ndecided, [q]uit this package
=><= k
```

Figure 6
Inspecting the inconsistent hypotheses

Figure 6 shows the inspection of the hypotheses that are responsible for the contradiction. Although the context under consideration is the set {wff6, wff13, wff15, wff28, wff31, wff89} only the hypotheses represented by wff13, wff15 and wff89 were used in the derivation of the contradiction and thus they are the only ones whose change will restore consistency. The

SWM system guarantees that removing *exactly* one of them will generate a context which is not known to be inconsistent. We keep the hypothesis concerning the statement of the puzzle (wff89) and discard the hypotheses concerning the women's ages (wff13 and wff15); We also enter new hypotheses concerning the women's ages (Figure 7).

```
The following (not known to be inconsistent) set of hypotheses was also part of the
context where the contradiction was derived: (wff31 wff28 wff6)
Do you want to inspect or discard some of them?
=><= n

Do you want to add some new hypotheses?
=><= y
Enter an hypothesis using SNePSLOG
=><= age(Bea,o30)
Do you want to enter another hypothesis?
=><= y
Enter an hypothesis using SNePSLOG
=><= age(Deb,o30)
Do you want to enter another hypothesis?
=><= n
```

Figure 7
Adding new hypotheses

After resolving the contradiction the inference resumes (Figure 8). In this case there is no further contradiction detected and SNeBR reports that Freeman will marry Deb and will not marry Ada, Bea, Cyd nor Eve.

I wonder if $marry(Freeman,who)$
holds within the BS defined by the context (wff14 wff16 wff6 wff28 wff31 wff89)

I know $age(Deb,o30)$

I know $age(Bea,o30)$

I know $worker(Deb,teacher)$

I know $worker(Ada,teacher)$

since $_{22}^{2}(age(Deb,o30),worker(Deb,teacher))$

I infer $marry(Freeman,Deb)$

since $age(Bea,o30)$ and $age(Deb,o30)$

I infer $_{10}^{0}(age(Eve,o30))$ $_{10}^{0}(age(Cyd,o30))$ $_{10}^{0}(age(Ada,o30))$

since not $_{22}^{2}(age(Eve,o30),worker(Eve,teacher))$

I infer $_{10}^{0}(marry(Freeman,Eve))$

Figure 8
Freeman will marry Deb
Eve, Cyd and Ada are not over 30
Freeman will not marry Eve

CONCLUDING REMARKS

We discussed SNeBR, the belief revision system used by SNePS; briefly described some of the concepts of the logic that underlies SNeBR; and showed an example. The example presented was obtained from an actual run just by slightly changing the syntax of the propositions.

SNeBR is implemented in SNePS, a powerfull knowledge representation system. A distinguishing characteristic of SNeBR is that it is based on a logic designed with the goal of sup-

porting belief revision systems. SWM associates each proposition with all the hypotheses used in its derivation and with all the hypotheses with which it is known to be incompatible The SWM formalism guarantees that (1) The origin set of a supported wff contains every proposition that was used in its derivation. (2) The origin set of a supported wff only contains the hypotheses that were used in its derivation. (3) The restriction set of a supported wff records every set known to be inconsistent with the wff's origin set. (4) The application of the rules of inference is blocked if the resulting wff would have an origin set known to be inconsistent.

In SNeBR, propositions are represented by SNePS network nodes and are indexed by (linked with) the hypotheses in their origin set and the sets in their restriction set.

The queries to SNeBR are associated with a context, the network retrieval function only considers the propositions in the BS defined by that context. When a contradiction is detected, after selecting one hypothesis (or several hypotheses) as the culprit for the contradiction, the "removal from the network of all the propositions depending on such hypothesis (hypotheses) is done just by dropping it (them) from the context being considered. Afterwards these propositions will no longer be in the BS under consideration and thus will not be considered by SNeBR.

ACKNOWLEDGEMENTS

Many thanks to Gerard Donlon, Donald McKay, Ernesto Morgado, Terry Nutter, Bill Rapaport and the other members of the SNePS Research Group for their comments and criticisms concerning the current work.

REFERENCES

Anderson A. and Belnap N., **Entailment: The Logic of Relevance and Necessity**, Vol.1, Princeton University Press, 1975.

deKleer J., "Choices without Backtracking", **Proc. AAAI-84**, pp.79-85.

Doyle J., "A Truth Maintenance System", **Artificial Intelligence**, Vol.12, No.3, pp.231-272.

1979.

Kaplan R.M., "A Multi-processing Approach to Natural Language", **Proc. National Computer Conference 1973**, pp.435-440.

Martins J., "Reasoning in Multiple Belief Spaces", Ph.D. Dissertation, Department of Computer Science, SUNY at Buffalo, May 1983.

Martins J. and Shapiro S., "Reasoning in Multiple Belief Spaces", **Proc. IJCAI-83**, pp.370-373.

Martins J. and Shapiro S., "A Model for Belief Revision", **Proc. of the Non-Monotonic Reasoning Workshop**, pp.241-294, AAAI, 1984.

Martins J. and Shapiro S., "A Logic for Belief Revision", forthcoming.

McAllester D., "An Outlook on Truth Maintenance", A.I. Lab., M.I.T., AI Memo 551, 1980.

McKay D.P. and Martins J.P., "SNePSLOG User's Manual", SNeRG Technical Note No.4, Department of Computer Science, SUNY at Buffalo, 1981.

McKay D.P. and Shapiro S., "MULTI - A LISP Based Multiprocessing System", **Proc. 1980 LISP Conference**, pp.29-37.

McKay D.P. and Shapiro S., "Using Active Connection Graphs for Reasoning with Recursive Rules", **Proc. IJCAI-81**, pp.368-374.

Shapiro S., "The SNePS semantic network processing system", in **Associative Networks**, N.V.Findler (ed.), Academic Press, pp.179-203, 1979a.

Shapiro S., "Using Non-standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network", presented at "Current Aspects of AI research", a seminar held at the Electrotechnical Laboratory, Tokyo, August 27-28, 1979b.

Shapiro S., Martins J.P. sand McKay D.P., "Bi-directional Inference", **Proc. of the Fourth Annual Conference of the Cognitive Science Society**, pp.90-93, 1982.

Shapiro S. and Wand M., "The Relevance of Relevance", Technical Report No.46, Computer Science Department, Indiana University, Indiana, 1976.

Steels L., "The Definition and Implementation of a Computer Programming Language based on Constraints", A.I. Lab., M.I.T., Technical Report TR-595, 1980.

Summers G., **Test your Logic**, Dover Publications, 1972.

Wand M., "The Frame Model of Computation", Technical Report N.20, Computer Science Department, Indiana University, Indiana, 1974.

3 DISTRIBUTED PROBLEM SOLVING

Report submitted by:
Susan E. Conry
Robert A. Meyer

Electrical and Computer Engineering
Clarkson University
Potsdam, NY 13676

and
Janice E. Searleman

Mathematics and Computer Science
Clarkson University
Potsdam, NY 13676

TABLE OF CONTENTS

I. Introduction 246

II. Application Domain Description 247
Organization of the DCS 247
System Control of the DCS 248
Performance Assessment 249
Fault Isolation 249
Service Restoral 250

III. System Architecture 251

IV. Knowledge Representation 254
Domain Knowledge 254
Design of the Knowlege Base 256
Graphics Interface 257

V. Problem Solving Strategies 260
Performance Assessment 260
Service Restoral 260
Fault Isolation 262

VI. Progress Review and Future Directions 264

VII. Bibliography 265

VIII. Appendix 3-A: A Shared KNowledge Base for Independent Problem Solving Agents 267

1. INTRODUCTION

This report describes the research efforts during the past year at Clarkson University under the Artificial Intelligence Consortium contract with the Rome Air Development Center. Our work is in the area of distributed artificial intelligence. Distributed AI is the study of systems involving the cooperation and coordination of a group of loosely coupled knowledge based component systems working together to solve common problems. Problem solving activity may be distributed for several different reasons. In some instances, distribution is used to provide a speed enhancement; in others, distribution is along functional lines with each problem solving agent having a particular specialization. When the problem domain is also distributed, a natural environment for distributed AI is created, and often such problems are not amenable to a centralized, non-distributed approach.

Our interest is in systems which are naturally distributed as the result of a distributed problem domain. Specifically, the system currently under study is both functionally distributed and geographically distributed (as the result of a geographically distributed problem domain). The long term goals of this project are to answer fundamental questions about distributed problem solving activities in such systems. For example, what are appropriate cooperation paradigms and how is the selection of these paradigms influenced by the type of problem solving activity required? How does the organizational structure of the distributed problem solver impact the ease of system construction and performance in problem solving? What forms of knowledge representation are suitable for sharing among the various problem solving agents?

In order to investigate these questions we have selected an application domain in which to study our ideas about distributed problem solving. This problem domain is the control of large communications networks. Believing that experimental testing of these ideas is a necessary step toward producing significant contributions to the field, we are currently developing a testbed for simulating the communications system control environment, and testing distributed problem solving systems.

The next section describes the specific problem domain we are using in this research. Much of our effort this past year has been in the analysis of the problem domain in order to gain a thorough understanding of its operation and to gather the basic knowledge needed to identify the problem solving tasks. There still remains the need to acquire detailed knowledge necessary to construct a complete knowledge base for problem solving activities. The third section presents an initial system level architecture based on the results of the domain analysis. Section four discusses knowledge representation for a distributed, shared, knowledge base. We give brief outlines of candidate high level problem solving strategies in section five. This material is in a very preliminary state and subject to great revision as the work continues in the next year. Finally in section six we review the research objectives and our progress in meeting these objectives, and outline the directions for future research.

II. APPLICATION DOMAIN DESCRIPTION

The application domain of interest for this research effort is the monitoring and control of large communications systems. Maintaining reliable communications under a wide variety of operating environments is vital to the preservation of both our national security and world peace. It is an important problem to the Department of Defense, and as we will discuss in the paragraphs which follow, it provides a rich set of problems for research in distributed problem solving.

In our studies we have concentrated on the Defense Communications System (DCS), and especially on the European theater. The DCS is a highly complex system consisting of tens of thousands of circuits interconnecting users at more than 300 sites world-wide. We have chosen the European theater for several reasons. The DCS network structure in Europe is particularly interesting for the study of distributed problem solving paradigms. It consists of a large number of sites (about 200) which are interconnected in an irregular structure. It is currently controlled by close cooperation and coordination among a group of highly skilled human controllers distributed throughout the system. The variety of transmission media and communications equipment in use give rise to the need for sophisticated problem solving tools to assist these human operators in providing the best possible control of the system.

The size, complexity, and near constant state of change of the DCS makes it unwieldly for direct incorporation into our investigations. Instead, the goals of this research program will be better served by using a simplified model of the DCS which incorporates those characteristics important to system control. This section describes the organization of the DCS and the tasks involved in system control. Emphasis is placed on those system features and aspects of problem solving activity relevant to our model of the DCS.

Organization of the DCS

The DCS is a large, complex communications system consisting of many component subsystems. It provides the long-haul, point-to-point, and switched network communications needed by the DoD. A careful analysis of the DCS reveals that the organization of the DCS must be viewed from a multidimensional perspective. For example, all DCS facilities may be divided into one of two groups: either DoD-owned or DoD-leased. As a general rule, the majority of DCS facilities in the continental U.S. are leased, whereas the majority of facilities overseas are owned and operated by the DoD.

The DCS may also be viewed as a layered organization consisting of three basic layers: transmission facilities, circuits and networks. Each of these layers may be further subdivided into component subsystems. Transmission facilities may be either terrestrial or satellite. Terrestrial transmission is based on either analog or digital channels, multiplexed into groups or digroups, and then into supergroups which are transmitted over communications links from one station to another. The most common transmission medium used is line-of-sight (LOS) microwave; however, there are also tropo-scatter, fiber optic, and cable links used. Satellite transmission facilities are also used, primarily for transoceanic links. Currently we have not studied satellite links, but we do intend to investigate the impact of satellite links in our future studies.

The transmission facilities form the backbone structure over which the second layer, consisting of circuits and trunks, is built. The European theater of the DCS consists predominately of dedicated circuits between users. These circuits may traverse several stations following fixed paths. There are a number of key data items which are associated with each individual circuit or trunk, and which are important in the performance of system control. These include the user priority level, the restoration priority, and the quality of service required.

Networks form a third layer of the DCS organization. There are three general categories of networks: voice switched, data switched, and dedicated or special purpose networks. These networks rely on trunks to provide the interswitch connectivity. The voice networks are AUTOVON, AUTOSEVOCOM (a secure voice network), and DSN (known as the European Telephone System or ETS in the European theater). These networks provide circuit switched voice connections among subscribers. The data networks include DDN, AUTODIN, and I-S/A AMPE. These networks are in a period of evolution from the older AUTODIN style network to the modern, packet switched DDN style network. We intend to incorporate the principal features of DDN into our model because we believe this network model is clearly the more significant for the future. Special purpose or dedicated networks are not considered at this time.

Yet another perspective of the DCS is equipment oriented. The DCS consists of a very large inventory of communications equipment, such as modems, multiplexers, radios, switches, etc. Each equipment item has certain distinguishing characteristics including its function within the overall system, its status signals (which may be monitored and made available to system controllers), and its control capabilities (which provide the mechanism for implementing desired control actions on the system). Knowledge about equipment is vital to problem solving agents attempting to control the system, and cuts across the layered organization described above. For example, a particular multiplexer may be a part of a transmission facility, as well as a part of one or more circuits, and a part of one or more networks.

The final dimension along which the DCS may be analyzed is its organization for monitoring and control. Currently the DCS system control function is almost entirely manual, and is highly fragmented. Each new network, or transmission subsystem incorporated into the DCS has included its own control system. As the DCS evolves to a modern, digital communications system, with automated control systems, it has become increasingly important to integrate these various controls. In the next section we discuss the system control problem. Our view of DCS system control is based on our understanding of the future directions system control for the DCS will take.

System Control of the DCS

System control is defined [12] as the process "... which ensures user to user service is maintained under changing traffic conditions, user requirements, natural or manmade stresses, disturbances, and equipment disruptions on a near term basis." System control incorporates five major functions: facility surveillance, traffic surveillance, network control, traffic control, and technical control. Each of these functions will be described in more detail and related to specific problem solving activities in the paragraphs which follow.

DCS system control is to be organized in a five level hierarchical structure. Beginning at the lowest level and moving up, each level in this hierarchy represents a broader view of the DCS, a larger geographic area, a greater responsibility and a higher authority. Level 5 (the lowest level) represents stations or facilities at which either a technical control or patch and test capability exists, or an access switch exisits, or an earth terminal for a satellite link exists. Level 4 represents either a major technical control facility or nodal switch. Level 3 represents a subregion control center (SCRF). Level 2 corresponds to theater level control and may be either an area communications operations center (ACOC) or alternate ACOC. Level 1 is the worldwide Defense Communications Agency Operations Center (DCAOC). For the purposes of our research, we are concerned with level 3 and lower levels. These are the levels most closely associated with the real time or near real time control of the system. Within the European theater approximately 13 SRCFs are expected to be established. Thus, it is at this level (level 3), or lower, that the need for cooperative problem solving is likely to be the greatest.

Three distinct problem solving activities have been identified within the five major functions of system control. We refer to these activities as performance assessment (PA), fault isolation (FI), and service restoral (SR). A general task description for each of these is given below and related back to one or more of the five functions of system control.

Performance Assessment (PA)

Performance assessment may be viewed as a problem in data interpretation and situation assessment. Since data is available only on a distributed basis, coordination must take place among the PA agents in order to arrive at a coherent view of the state of the communications system. The facility surveillance and traffic surveillance functions of system control are included within the PA activity. Real time equipment, transmission network, and traffic data are measured and collected to provide the controller with the information needed to determine the status of the transmission system and facilities, the quality of communications circuits and network performance. Trouble reports from users are also significant inputs to this activity.

The goal of PA is to formulate a local view of system status and performance, and to identify as quickly as possible the impact of any observed deviations from normal operating conditions. The PA agent is responsible for determining the need to invoke either fault isolation and/or service restoral agents. Since few problems are likely to be localized within the area of responsibility of a single SRCF, the PA agent must also communicate with similar agents in neighboring areas to arrive at a consistent assessment of status throughout the system.

Fault Isolation (FI)

The fault isolation task is a diagnostic activity. It is concerned with identifying the specific cause and location of faults within the communications system. The term "fault" is used in a very broad sense to mean either a complete outage of service or a degradation in quality or performance. The FI agent responds to reports of known or suspected faults

determined by PA. Much of the same data available to PA is also used in the FI activity, however the analysis is carried out in greater depth, and the cause-effect relationship is emphasized. In some instances the immediate results of FI may be inconclusive and will require additional testing to resolve ambiguities in the data.

The FI agent incorporates the in-depth analysis aspects of facility and traffic surveillance as well as the testing aspects of technical control. Coordination and cooperation with similar agents at intermediate or distant end stations involved in a faulty link, trunk, circuit, or network are often necessary to determine the cause and location of a fault.

Service Restoral (SR)

Service restoral is a plan generation activity which recommends a set of specific control actions needed to restore user service. These actions may involve alternate routing of trunks or circuits, switch control (such as code cancellation, code blocking, modification of routing tables, etc.), or transmission system configuration control (such as reallocation of equipment, use of backup or spare equipments, etc.). The network control, traffic control, and technical control functions are encompassed in the SR activity.

III. SYSTEM ARCHITECTURE

In this section, we describe the overall architecture of a communication system such as the DCS with the incorporation of an intelligent distributed problem solving system. This discussion addresses three primary facets of the architecture: the role of the intelligent software in the context of the overall system, the structure of the intelligent system residing at each node, and the overall structure of the distributed problem solving system.

Any deployed distributed problem solving system would be most accurately viewed as an intelligent assistant to the system controllers in the field. It would perform the tasks of filtering the data received, analyzing and interpreting it as well. Based on these interpretations, it would suggest alternative restoral plans and advise the humans as to which equipments are probably malfunctioning and what control actions might be appropriate. The human operator would have the responsibility of directing service restoral, dispaching service personnel, and initiating control actions. In addition, the human retains the privilege of preempting the system at any time. As an intelligent assistant, the system would relieve the controller of many tedious tasks. It would also provide a vehicle for training activity that can be utilized by new personnel.

Distributed problem solving systems are often viewed as being comprised of a network of loosely coupled agents which cooperate in solving a problem. In the context of communication systems, each locus of control activity is also a site where a node of the problem solving network resides. As the discussion in the previous section indicates, several modes of problem solving are active concurrently. Performance assessment is an ongoing task, while fault isolation and restoral tasks may be initiated and execute at the same time. We have chosen a node architecture that represents a decomposition of nodal problem solving activity into its three primary tasks. Each of these tasks appears to be relatively independent of the others, yet each requires access to the same base of knowledge concerning the status of the communication system and its expected behavior.

The structure of the testbed for distributed problem solving reflects distribution in two dimensions. At one level, the system is seen as a number of complex agents that operate in a loosely coupled fashion to solve the problem of controlling the communication network. At another level, the system can be regarded as a group of relatively independent distributed problem solving systems operating in a loosely coupled fashion. One of the systems is composed of the group of fault isolation agents. The fault isolation agent at each node cooperates with its counterparts at other nodes in solving the fault isolation problem for the communication system. In a similar fashion, the service restoral and performance assessment agents in the testbed can be regarded as distributed problem solving systems in their own right.

The specific structure appearing at each node is shown in Figure 1. As is indicated, there are three primary problem solving modules at the node level: Fault Isolation, Performance Assessment, and Service Restoral. Each of these modules requires access to the same knowledge about the structure and expected behavior of the network being controlled. Since each local agent performs functions that may have impact on the problem solving activity of the others, a blackboard structure located in each node is used to post tentative

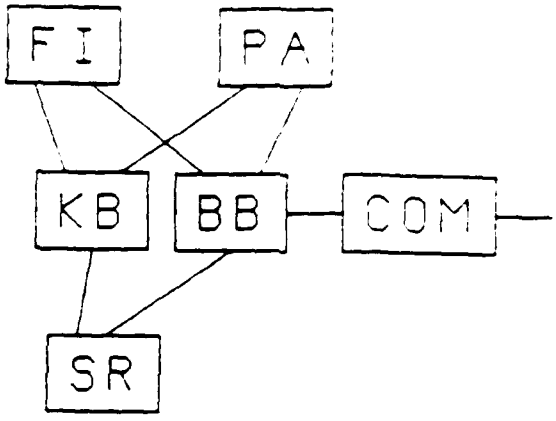


Figure 1

hypotheses and partial solutions as problem solving progresses. This local blackboard also may serve as a structure where the problem solving activity may be monitored and controlled through the posting of intermediate goals. One agent in the diagram has the responsibility of handling communication between a node and its neighbors in the problem solving system. This communication is necessary because no single node has a complete and accurate view of the system state. It is also required to solve problems in a distributed fashion.

In section V of this report, we discuss high level strategies for performing the major tasks involved in system control. The kinds of results and the modes of problem solving activity required to implement strategies such as these suggest a local blackboard structure which is segmented by content. Each agent has access to the local blackboard but is most interested only in certain portions of it. An example of the type of segmentation envisioned is found in Figure 2. In this figure, the segmentation and primary areas of interest of each agent are shown.

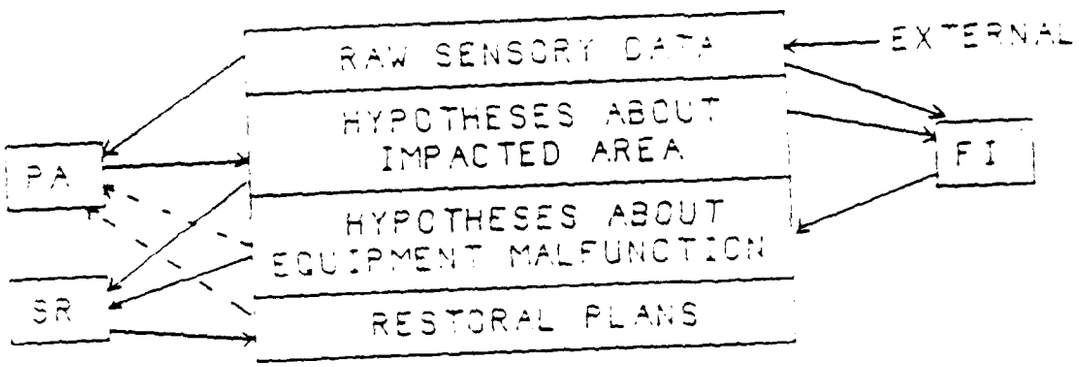

RAW SENSORY DATA
EXTERNAL
HYPOTHESES ABOUT IMPACTED AREA
PA
FI
HYPOTHESES ABOUT EQUIPMENT MALFUNCTION
SR
RESTORAL PLANS

Figure 2

IV. KNOWLEDGE REPRESENTATION

We have designed a knowledge representation scheme which supports the problem solving activities performed at a single node in the monitoring and control of the communications network. The local node knowledge base contains information about the structure, function and current state of the network, and will be shared by the various problem solving agents at that node.

Domain Knowledge

The knowledge necessary to perform the system control functions described earlier falls into three basic categories. First there is equipment configuration knowledge which describes the physical structure of the network including specific equipments at each site, connectivity information, and a description of each kind of equipment. Secondly, equipment state knowledge contains information about the operating status of each piece of equipment, such as current alarms and other information gained from the equipment sensors, as well as conclusions to be made about this sensory data. Finally, communications path knowledge represents the specific equipments and links involved in a path connecting two users and more general knowledge about how to handle interrupted paths.

Equipment configuration knowledge is largely topological in nature, and describes the physical communications network. Knowledge about this network is hierarchical in nature, where at the highest level stations are interconnected via links, and collections of stations are grouped into subregions which correspond to geographic areas. Within each subregion, one station is designated as the current SRCF or SubRegion Control Facility, and is responsible for the control decisions made for the stations within its subregion, or area of responsibility. Each station corresponds to a node or site in our knowledge-based problem solving system. The knowledge base therefore contains declarative knowledge concerning each site in the network, including its name, geographic location, what subregion it is contained in, whether it is the SRCF for its subregion, and its current operating status. In addition, each link in the network is also represented, with information concerning which sites correspond to the link's two endpoints, and the media type and capacity of the link. This level in the hierarchy is illustrated in Figure 3.

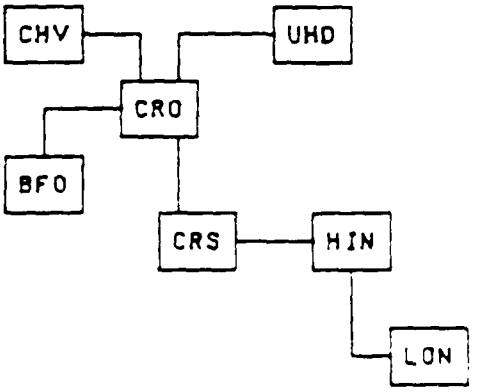


Figure 3

At the second level in the hierarchy, within each site there is a collection of interconnected equipments. The physical topology is similar to the top level network, but at this level specific pieces of equipment are interconnected by various kinds of arcs. A typical equipment configuration for a site is given in Figure 4. Types of equipment include radios, second level multiplexors (MUX), first level MUXes, and digital patch and access systems (DPAS). Interconnecting these equipments, and connecting equipments with users, are supergroups, digroups, and channels. The equipment configuration knowledge within each site also forms a hierarchy. A radio within a site may be connected to a second level MUX or to another radio at the same site via a supergroup. Each second level MUX is connected either to a first level MUX or to a DPAS via a digroup. Finally, each first level MUX is connected to a user via a channel. The endpoints of intersite links correspond to specific radios within the connected sites.

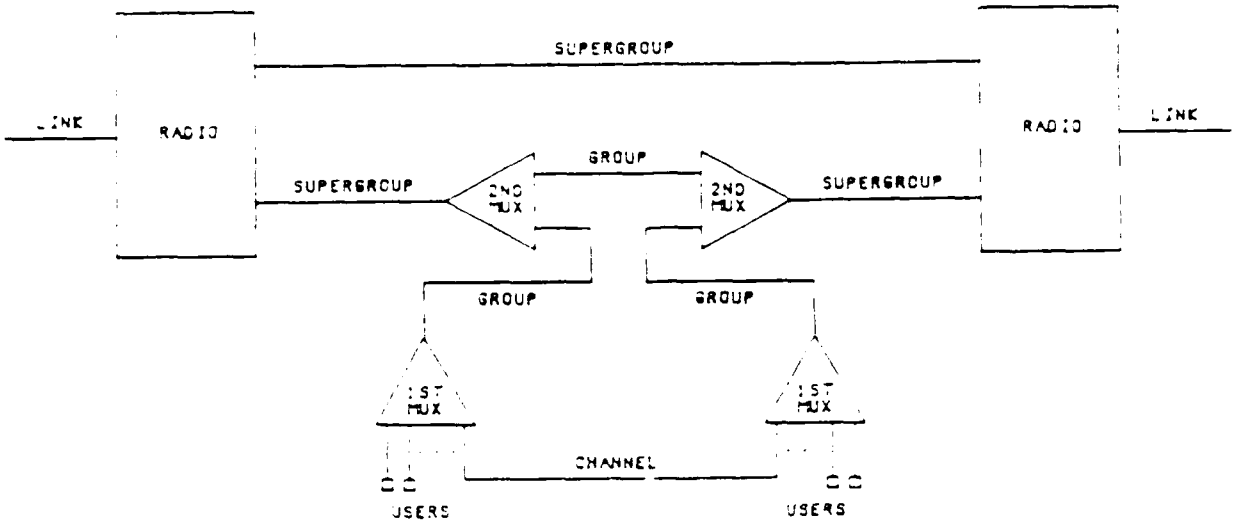


Figure 4

In addition to the knowledge described above concerning specific pieces of equipment and their connectivity, equipment configuration knowledge also contains more general information on each item in the hierarchy; that is, generic information on subregions, stations, links, as well as kinds of equipments and connections among them is represented in the knowledge base. For instance, generic information common to all radios of the same type includes the type of alarm signals which may occur, specific actions to take in the advent of such alarms, and operational parameters for that kind of radio.

Equipment state knowledge is also necessary for system control. Whereas equipment configuration knowledge describes the physical connectivity of the network, equipment state knowledge contains information concerning the current operational status of each piece of equipment. This includes any current alarm signals for the equipment, but also includes expected behavioral characteristics for each kind of equipment, and any deviations from this norm which a specific piece of equipment might have. This data is used by the

Performance Assessment agent as it analyzes the current state of the communications network.

Finally, communication path knowledge reflects the current user-to-user connections in the network. The primary unit of telecommunications service carrying message traffic between two locations is known as a communications channel. The three telecommunications connectivity entities are circuits, trunks and links, which carry the transmitted signals in a communications channel. A circuit is a path between two end-users, and consists of a sequence of adjacent nodes and arcs such that no node or arc appears more than once in the path. A node corresponds to a location where a communications signal may be originated, manipulated, or terminated, and an arc consists of the set of communications channels between two adjacent nodes. Examples of nodes include end-user points at some location, drop-and-insert points which correspond to intermediate nodes in a communications channel, and PTT-pickup points where a communications channel is transferred from a DCS controlled transmission facility to a common-carrier transmission facility. A trunk is a single communications channel between two or more nodes, and may itself be channelized; a trunk or any of its channels may carry a single circuit or another trunk, and the signals at the initial node and the terminating node of the trunk are in the same form. Finally, a link is as described above in the equipment configuration knowledge. It is a transmission facility, such as a cable or microwave radio system, connecting two adjacent nodes, and may be channelized. Also, the topographic terms of node, arc, and path are related to geographic terms location, link and route. That is, a route is a sequence of transmission facilities traversed by a communications channel, and so can be described by a sequence of locations and links. Also, communications path knowledge is time-dependent, as circuits and trunks are associated with a particular configuration only for a finite period of time, and may be alternately routed if necessary.

Design of the Knowledge Base

A frame-based system is used to embody the equipment configuration knowledge and equipment state knowledge. This allows us to easily describe specific pieces of equipment, including its current state, where it is located, and to what it is connected, and also to describe the specific path components involved in each connection. In addition, generic information about each kind of equipment and path component is easily captured in frames, which are also used to provide default values and information on actions to be taken during problem solving activities involving items of that type.

Figure 5 gives an example of the equipment configuration of two sites connected by a link, and Figure 6 shows how this knowledge is stored in the knowledge base. In the latter figure, a semantic net shows the relationships among the various frames corresponding to specific equipments and arcs, as well as generic frames, and closely follows the hierarchy inherent in the system. There are instances of frames corresponding to specific sites, links, radios, supergroups, level 2 muxes, digroups, level 1 muxes, channels, and users, each of which has an "isa" link to generic frames for those items.

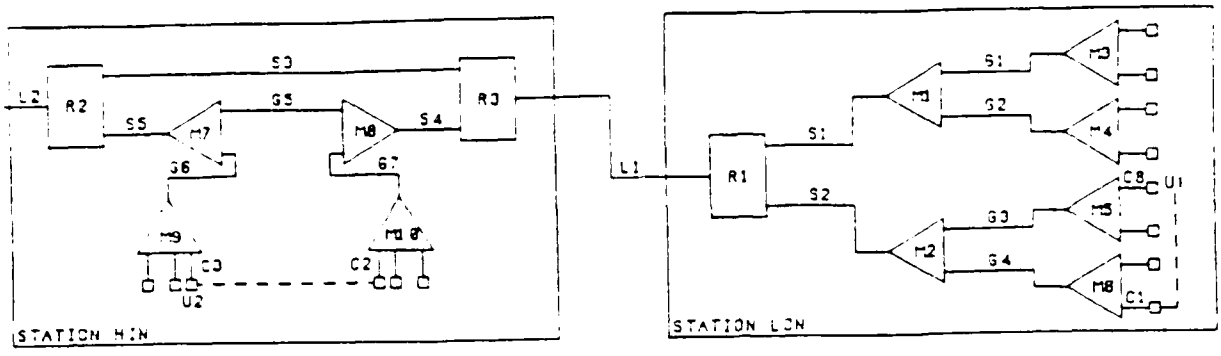


Figure 5

This knowledge base is currently being implemented in ZetaLISP on a Symbolics 3670. Object-oriented programming techniques are being employed to describe the abstract data types needed, and to provide data encapsulation. This provides the modularity and flexibility needed in a developing system, and message passing corresponds well to communication in a distributed system. In addition, since many of the objects have homogenous features, the implementation makes extensive use of the flavor facility. For example, there are many common operations to be performed on equipments such as radios, multiplexors and switches. Similarly, path components such as links, supergroups, digroups, etc. have common properties. Finally, the window system provides a convenient user interface for creating and modifying the knowledge in a communications network.

The choice of LISP was influenced by several factors in addition to its availability on the LISP machine. PROLOG, for example, has many features which would be useful in representing such a system, such as the direct support of pattern-matching and backwards chaining. However, it was felt that there is inadequate control over the search mechanism, and that it would be difficult to deal with incomplete knowledge in PROLOG. Also, the support environment for developing large systems, including debugging facilities and the flavor examiner, is superior in ZetaLISP than to any currently available version of PROLOG.

Graphics Interface

We have designed and are currently implementing a program which essentially provides a user with a graphical network editor. That is, the user will have the ability to create, modify, and save communications networks. This facility operates on different levels. At the highest level, the user can draw a network consisting of sites and links by using the mouse to position a site or to identify the endpoints and type of a link. These sites and/or links can be added, deleted, named, or modified. The resulting network can be named, saved on a disk for future use, or a previously save network can be loaded. As the user enters this information, the program instantiates the appropriate flavors thus automatically building a knowledge base.

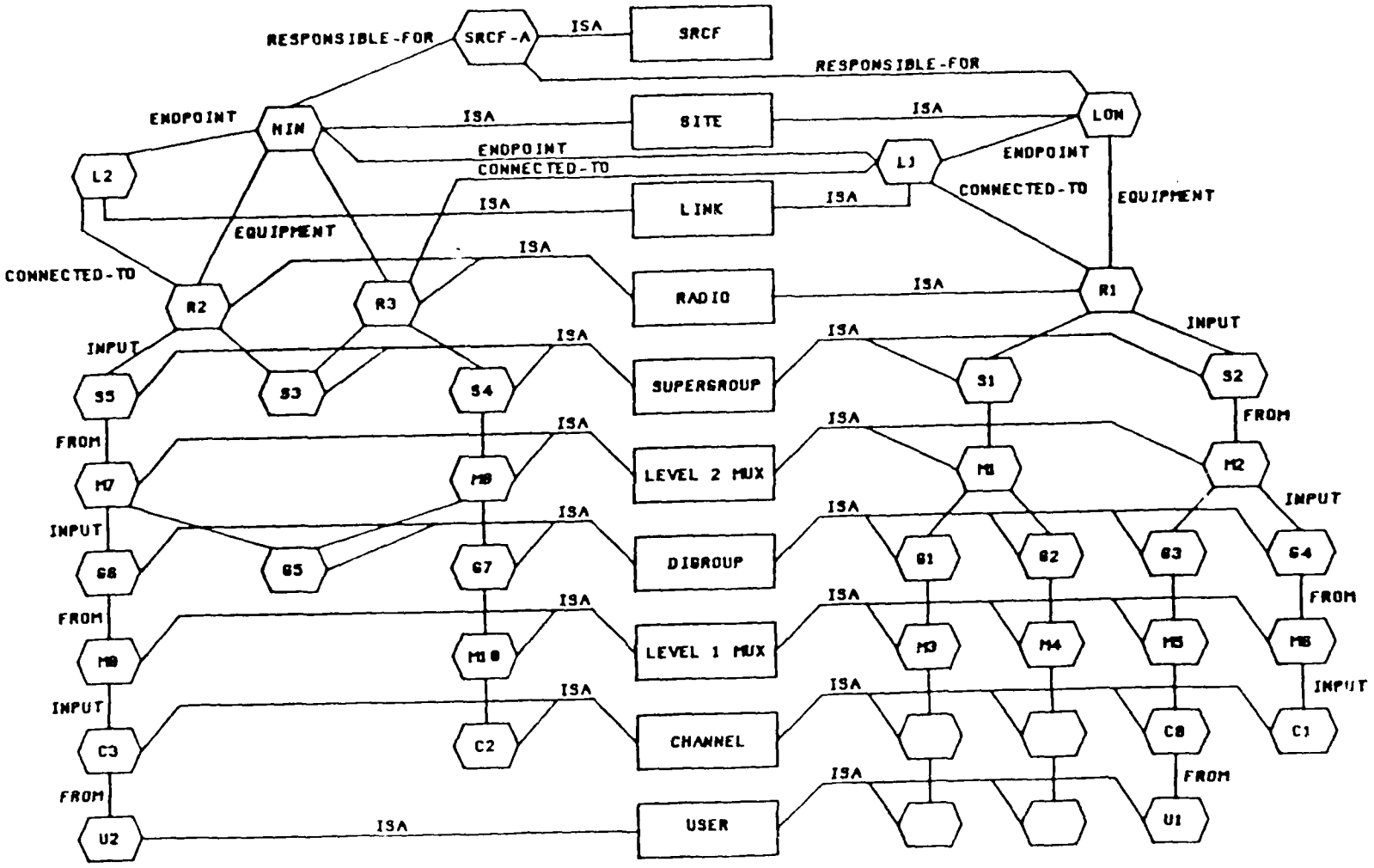

RESPONSIBLE-FOR
SRCF-A
ISA
SRCF
RESPONSIBLE-FOR
ENDPOINT
NIN
ISA
SITE
ISA
LON
ENDPOINT
L3
ENDPOINT
CONNECTED-TO
L2
ISA
LINK
ISA
CONNECTED-TO
EQUIPMENT
EQUIPMENT
ISA
CONNECTED-TO
R2
R3
RADIO
ISA
R1
INPUT
ISA
ISA
INPUT
S5
S3
S4
SUPERGROUP
S1
S2
FROM
ISA
ISA
FROM
M7
M8
LEVEL 2 MUX
M1
M2
INPUT
ISA
ISA
INPUT
G8
G5
G7
DIGROUP
G1
G2
G3
G4
FROM
ISA
ISA
FROM
M9
M10
LEVEL 1 MUX
M3
M4
M5
M6
INPUT
ISA
ISA
INPUT
C3
C2
CHANNEL
C8
C1
FROM
ISA
FROM
U2
ISA
USER
U1

Figure 6

For each node in the network, a mouse-sensitive region is created, thus allowing the user to select a site with the mouse and enter the second level of the network editor. This causes the window associated with that site to be made active, allowing the user to view the equipments and connections within that site. The facilities at this level are analogous to the top level. That is, equipments and arcs corresponding to supergroups, digroups, etc. which connect those equipments can be added or deleted by using the mouse to select and place pieces of equipment or to identify the endpoints of an arc. Again, as the user modifies the equipment configuration, the knowledge base is updated to reflect the changes.

V. PROBLEM SOLVING STRATEGIES

In this section, we briefly outline our current views as to appropriate avenues for development of some of the major problem solving agents resident at each node. There are three of these agents which are highly domain specific: performance assessment, service restoral, and fault isolation.

Performance Assessment

The performance assessment agent at each node has the responsibility of monitoring and filtering voluminous raw data. Its primary function (in the rudimentary system which is our first objective) are those of detecting when a fault has probably occurred and assessing the area impacted by a fault. Once this has been done, performance assessment may activate fault isolation to locate the source of the problem or problems and service restoral to generate appropriate restoral plans.

Performance Assessment must monitor three kinds of raw data: equipment alarm and status indicator values, quality of communication as reflected by user complaints, and system performance as indicated by traffic data. Any patterns in the raw data that are indicative of a fault must be recognized and handled appropriately.

Fortunately, the status and alarm data reported is only that involving changes in the status of equipment indicators. Some of the alarm states on specific pieces of equipment are themselves indicative of problems. Others must be interpreted in the context of the status at other equipments connected to one that is alarmed. We believe that the context of physical interconnection structure in the locality of the alarm plays a large role.

User complaints form a kind of trouble reporting that should be recognized and handled at once. Many of the user complaints may be confirming evidence of trouble that is also detected by the presence of alarms on various equipments. In order to minimize the amount of work done, handling of user complaints may involve placing a very high priority on interpretation of sensory data along the communication path assigned to the complaining user. This suggests that possibly one fruitful approach to the design of a rudimentary performance assessment agent would be rule based. The control strategy would be largely data driven, with user complaints dictating high priority areas of interest.

At this time our efforts are directed mainly towards assessment that takes active indications of faults as its primary input. Traffic data is also important to performance assessment, but it is significant in a negative sense. Trouble may be indicated if there is a lack of traffic in a given area. Although we recognize the importance of traffic data in a realistic model of this task, we have chosen to defer careful consideration of its use in problem solving. For the present, we are primarily concerned with interpreting and assessing the impact of raw status and alarm indicators and user complaints.

Service Restoral

In the context of this system, Service Restoral has the responsibility of determining recommended control actions needed to restore interrupted service

to users at different sites. One possible control action is often the use of an alternate routing while the fault is repaired. There are a number of "rules of thumb" and constraints that come into play in solving this problem. Our proposed approach to generating an "alt-route" plan would view this task as a distributed constraint satisfaction problem. We believe that this approach may provide greater flexibility with less complexity than a rule based approach. The reason lies in the number of variations that can arise, especially when multiple outages occur and multiple restoral plans are being generated.

The assumption is that a new route is not required unless one of two kinds of events has occurred: (1) service has been requested, so it is necessary to establish a new circuit between two sites, or (2) some trouble in the system has interrupted service between two users, so an alternative circuit must be found. Service Restoral activity is initiated in slightly different ways for each of these two cases, but the problem solving paradigm is the same in either case.

In rough terms, the problem of alt-route generation involves finding a new circuit between two users. If new service is being requested, the high level goal of establishing a circuit between site A and site B is inserted on the local blackboard of the controlling node for site A or site B. The service restoral agent at a site with that as its highest rated goal establishes a connection within its area of responsibility and passes the subgoal of finishing the task to its neighbors. The neighbors operate in a similar fashion: when the goal transmitted is the highest rated goal (locally), that portion of a possible circuit in the appropriate area of responsibility is proposed and further goals are transmitted to neighbors. In this way, a plan for establishing service is determined.

The scenario is slightly different in a situation involving restoral of service that has been interrupted. In this case, there should be hypotheses on a local blackboard regarding the channels, digroups, etc. that are affected by troubles, hence should not be used. Existence of these hypotheses causes a goal to be created: restore service to users who were using the affected resources. If this goal can be satisfied locally, there is no need to propagate it to neighboring nodes. If not, a partial plan is generated locally and subgoals are propagated to neighboring nodes. Aside from its initiation, the way in which plan generation is controlled is the same in either case.

At this time, we have identified three general categories of constraints: those which reflect that a resource is currently fully utilized (FULLY-UTILIZED constraints), those which indicate the circumstances under which a user may be preempted (PRIORITY constraints), and those which reflect existing areas of trouble in the network (NOTUSE constraints). The problem solving paradigm which is currently envisioned for alt-route generation is shown in the diagram below.

```
I       observe local BB hypotheses
        about affected resources

II      update constraints based on
        what resources cannot be used
        (NOTUSE constraints)

III     process highest rated subgoal
        (highest rated goal dependent on
           priority of circuit to be made
           and time the goal was posted)
        (goal is of the form connect X to Y:
         if there is a spare, assign it
         if not, preempt lowest priority
                 post goal of connecting preempted segment
                 assign the preempted resources)

IV      propagate results
          (send new bindings, goals, and circuit changes
             to neighbors
           update local knowledge base)

V       go to I
```

Fault Isolation

The task for this problem solving agent is to pinpoint the specific location of a fault within a network, a circuit, or the transmission media which carries the circuit. The trouble may be an outright failure of some transmission facility, or it may be the result of degradation in the quality of communications rendering a circuit useless. The first step in fault isolation is to recognize that a problem exists. Trouble recognition is a function of equipment sensors and alarms, user complaints, and quality control testing. The Performance Assessment module analyzes this raw data, quickly assesses the impact of the outage, and posts initial hypotheses concerning probable sources of the fault(s). This activates the Fault Isolation module, which then analyzes the equipment alarms and posted hypotheses, suggests diagnostic tests to help locate the problem, and deduces the faults which have occurred and the facilities causing them. This task may be complicated by the fact that the incoming sensory data may either be incomplete or it may contain conflicting information. For example, the failed equipment itself might not

generate an alarm. Also, the original problem may, and often does, generate extraneous or sympathetic alarms which then propagate throughout the network. Finally, multiple faults may generate a large collection of data which must then be sifted through and unentangled. Thus fault isolation must efficiently discard artifacts such as sympathetic alarms and determine which tests are necessary in order to isolate the trouble rapidly.

We believe that a rule-based forward chaining paradigm may be useful in solving the fault isolation problem. The rules are partitioned according to generic equipment type, and there are a fairly small number of rules for each kind of equipment which correspond to the kinds of alarms which may occur for that equipment. The antecedant for each rule consists of a possible alarm or pattern of alarms, and the consequent consists of an action to be taken, such as to post some message on the Blackboard, or to request more information or initiate a diagnostic test.

Rules such as those mentioned above embody shallow knowledge about the system. That is, they map patterns of symptoms to diagnostic conclusions. However, human experts performing the fault isolation task often resort to experience and using fundamental knowledge about the nature of the equipment in order to diagnose the problem more rapidly. Thus we envision augmenting a simple system to also employ such deep knowledge and to reason from first principles when necessary. The knowledge would need to contain information on the functionality and behavior of each component in the system, and a mechanism would be needed to control the reasoning process, and to either activate the shallow inference engine or the deep inference engine.

It is envisioned that a goal structure would be associated with fault isolation to aid in control of the agent's problem solving activity. Such a goal structure, together with partial hypotheses, could guide the search for a solution, assuming the existence of a high level goal such as "find the source of trouble". Subgoals of two types aid in satisfying the high level goal. A subgoal of the form, "Perform a test", requests more sensory data from the communications system. "What do you know" subgoals request information from neighboring nodes. Multiple activations of Fault Isolation by Performance Assessment must be handled carefully, recognnizing that several activation messages may be sent to Fault Isolation from Performance Assessment as a result of the same problem.

VI. PROGRESS REVIEW AND FUTURE DIRECTIONS

The major goal of this five year research project is to answer fundamental questions about distributed problem solving systems. We believe this goal can be achieved by the development of a testbed for conducting experimental studies on the performance of various distributed problem solving paradigms. The testbed embodies an application domain which is naturally distributed and which presents a challenging set of tasks for problem solving agents.

At the completion of the first year of this effort we have gained a sufficient understanding of the application domain to formulate a general model of the problems to be solved and to design a representation technique for the domain knowledge necessary to solve these problems. This representation technique has been partially implemented using an object-oriented programming methodology. The result of this effort is a graphics-based tool for the construction of a specific instance of a domain knowledge base. Our analysis of the application domain has identified three basic types of problem solving activity: (1) a data interpretation and situation assessment task, (2) a diagnostic task, and (3) a plan generation task. Each of these tasks must be conducted in a distributed environment. Candidate stategies have been developed for performing each of these tasks.

In the coming year our efforts will directed toward completing the basic testbed environment, and implementing initial versions of the problem solving agents. Another part of our work during the next year will be to acquire the detailed knowledge about the problem domain necessary to construct a fairly complete knowledge base. Our plans for knowledge acquisition include continuing study of the available documentation, interviews and discussions with experienced DCS system controllers, and actual visits to representative DCS sites.

BIBLIOGRAPHY

[1] R. Davis and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence, Vol. 20 (Jan. 1983) 63-109.

[2] R.G. Smith and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving", IEEE Transactions on Systems, Man, and Cybernetics, Vol. C-129, No. 12 (Dec. 1980) 1104-1113.

[3] D. M. Schutzer, "Concept of Control of a Military Digital Communications Network", ICC 76 Conference Record, (June 1976), 34-7 - 34-13.

[4] E. H. Shortliffe, Computer-based Medical Consultations: MYCIN, American Elsevier, New York, 1976.

[5] R. O. Duda, P. E. Hart, N. J. Nilsson, and G. L. Sutherland, "Semantic Network Representations in Rule Based Systems" in D. A. Waterman and F. Hayes-Roth (Eds.), Pattern Directed Inference Systems, Academic Press, New York, 1978.

[6] R. Davis, H. Schrobe, W. Hamscher, K. Weickert, M. Shirley, and S. Polit, "Diagnosis Based on Description of Structure and Function", Proceedings of AAAI-82, August 1982.

[7] M. S. Fox, S. Lowenfeld, and P. Kleinosky, "Techniques for Sensor-based Diagnosis", Proceedings of IJCAI-8, August 1983.

[8] T. Finin, J. McAdams, and P. Kleinosky, "FOREST - An Expert System for Automatic Test Equipment", Proc. First Conference on Artificial Intelligence Applications, (December 1984), 350-356.

[9] F. Pipitone, "An Expert System for Electronics Troubleshooting Based on Function and Connectivity", Proc. First Conference on Artificial Intelligence Applications, (December 1984), 133-138.

[10] D. Corkill and V. Lesser, "The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network", Proc. of IJCAI-8, August 1983.

[11] D. Corkill, "A Framework for Organizational Self-Design in Distributed Problem Solving Networks", PhD Thesis, Univ. of Mass., Amherst, Mass., 1983.

[12] Defense Communications Agency Circular 310-70-1 (DRAFT), Vol. I (March 1984), Vol. II (Oct. 1984), and Vol. III.

[13] Defense Communications Agency, "Baseline Conceptual Design For The DCOSS Data Base", July 1985.

A SHARED KNOWLEDGE BASE FOR INDEPENDENT PROBLEM SOLVING AGENTS

by
S. E. Conry, R. A. Meyer, and J. E. Searleman

Clarkson University
Potsdam, N. Y. 13676

Abstract

In this paper, we describe a knowledge representation scheme that is intended to be compatible with two modes of problem solving activity in a distributed problem solving environment. The application domain of interest involves the situation assessment, diagnosis, and planning necessary for monitoring and control of large communication systems.

Introduction

Our research is ultimately concerned with investigating cooperation paradigms for distributed problem solving in the context of an application domain that involves the monitoring and control of a large communication network. The end goal is one of developing tools that will assist the technical controller in the field under normal operating conditions and especially in times of stress.

Distributed problem solving is concerned with the activity of several problem solving agents in determining a solution to a problem. It has been characterized as the cooperative solution of problems by a decentralized, loosely coupled collection of problem solving agents [1,2]. In our application domain, problem solving activity is distributed in two distinct ways. There is physical distribution as a result of the geographical separation of the various communications nodal sites. At each node, there is a need for functional distribution to solve problems with distinct goals and differing problem solving paradigms. This paper describes the design of a knowledge base to be shared among the problem solving agents residing at a single node.

At each node there are several modes of intelligent behavior required. Examples include interpretation of large amounts of raw data in performance of situation assessment and planning functions. Each of these functions requires access to the same base of local node knowledge about the state, structure, and normally expected behavior of the network, so design of the common local knowledge base is of some importance. In this paper, we first discuss the application domain and the tasks that are currently performed by humans at the node level. Next we describe strategies for solving two major problems that must be addressed in performing these tasks. We then describe the common knowledge needed to perform each of these tasks and our design for a common knowledge base structure. Examples are given of the problem solving activity performed using this knowledge base in doing planning and diagnosis relevant to the monitoring and control of communication systems.

Application Domain: System Control Problems for the DCS

The Defense Communication System (DCS) is a world wide military communications network consisting of various transmission and switching facilities. System control for the DCS involves four categories of tasks: data acquisition, data analysis and assessment, decision making, and control action execution. These tasks support functional capabilities for dynamic resource allocation, system wide performance assessment and fault isolation, and restoral [3]. At the present time, the only tasks that have been automated are those of data acquisition and execution of a few very low level control actions. Our research is directed towards development of intelligent systems to support data analysis and assessment and decision making activity. In this section, we give a brief summary of the organization and operation of the DCS.

The DCS consists of a variety of transmission, switching, and terminal facilities. We refer to any single such facility as a *station* or *site*. System control activities are performed at a subset of these sites which we will refer to as *nodes*. The tasks performed at these nodes include performance assessment, service restoration, and coordination of activities with other nodes.

At each node, data is collected locally at the node and via remote sensors at sites "near" the node in its area of responsibility. This data

This research was supported by the United States Air Force Rome Air Development Center under contract F30602-85-C-0008 through the Northeast Artificial Intelligence Consortium.

CH2225-1/85/0000/0178$01.00 © 1985 IEEE

is collected from a wide variety of sources, and is both voluminous and potentially inconsistent. It is used to assess the operating status and performance of the network. This is the performance assessment task. The data is also utilized to detect and localize equipment failures and link or station outages in doing fault isolation and diagnosis. In addition, decisions must be made at the node level (as well as at the system wide level) as to whether or not some or all traffic should be rerouted in response to a particular problem. Even if a first decision is not to reroute traffic, plans for rerouting must be developed in response to changing conditions so that they can be implemented rapidly should the situation require immediate action. Although the scope of this research is presently focused on the aspects of system control mentioned above, the DCS as a problem domain incorporates a number of other tasks as well. Some of these are switching and congestion control.

The problems involved in the monitoring and control of a military communications system are significantly different from those faced by commercial carriers in this country. The single most important distinction is found in the fact that some of the traffic is vital to the national security, and that it becomes even more essential that this traffic not be interrupted in times of crisis. This factor is exacerbated by the relatively sparse connectivity of the network. For example, in the European theater some 33% of all stations are directly connected to only one other station and another 40% have direct links to only two other sites. The problem is further complicated by the presence of various levels of priority for users. High priority users will preempt those of lower priority. All of these facets of system control make it a problem in which assessment and decision making must be done very quickly - as close to real time as possible. These functions must also be performed under stressful conditions when the integrity of much of the system may be in question.

Intelligent Systems for Fault Isolation and Restoral

There are two major system control tasks with which we have been most concerned. One is the problem of detecting a fault and locating the equipment causing the trouble (fault isolation). The other is the problem of generating restoral plans that would restore service, especially to high priority users, in the case of service interruption (restoral).

These problems have been studied extensively in the context of conventional algorithmic approaches, but the DCS is a problem domain for which adequate conventional algorithms are difficult (if not impossible) to design. Examples of factors which suggest the need for intelligent systems are: 1) the state of local knowledge about global network status is often not accurate because it is not up-to-date; 2) there is ambiguity in the meaning of monitor point alarms; 3) human technical controllers employ heuristic problem solving techniques and rely on their past experience, and 4) coordination among technical controllers at different sites is usually necessary to resolve discrepancies.

Fault Isolation

In the context of the DCS, a fault is the failure of some function. As a result of a fault (or some combination of faults), a set of alarms may be generated. In addition, degradation in the quality of communications may make a circuit useless. The fault isolation task can be stated briefly as follows: given a set of alarms and or user complaints, deduce the faults that have occurred and the equipments causing them.

The basic problem solving paradigm for solving this problem is a standard one in expert systems: observe the pattern of alarms and complaints; hypothesize possible sources of the trouble; communicate with other nodes if necessary to confirm or disconfirm the hypothesis, recommend corrective action.

Two factors act to decrease the certainty factor associated with a particular suggested solution. One is the presence of conflicting sensory data and extraneous or sympathetic alarms. The other is the observation that sensory data is often incomplete. There may be no alarm from the failed equipment itself, so the information available may not be sufficient to exactly pinpoint the trouble. For this reason, it is necessary to incorporate a measure of confidence that the equipments identified as faulty are indeed faulty. This sort of thing has, of course, been done before in rule based expert systems [4,5]. We envision a task specific knowledge base which enables a fault isolation agent to quickly discard artifacts such as sympathetic alarms and determine which tests should be performed to isolate the trouble rapidly. Examples of related work on fault detection and isolation in hardware include the work described in [6-9]. Our work in fault isolation is related most closely to that found in [6]. We incorporate a causal model of system components to relate observed system behavior to candidates for specific failure mechanisms. The difficulty arises because this model is not a simple formal mathematical model. Instead, it is a complex model which involves a large number of variables, many of which are not readily measured.

Restoral

An outage occurs if communications service between two or more users is interrupted. Outages may result from equipment failures, natural disasters, adverse atmospheric conditions, natural or intended interference, or physical conflict. When an outage is detected and is estimated to continue for more than a brief time (e.g. 15 minutes), then some or all of the interrupted services should be restored. Restoral means finding alternative paths or routes which do not traverse the affected equipments, links, stations,

etc. At the present time emergency restoral is done in accordance with prepared plans which are generated in advance and based on single failure assumptions.

Routing problems in networks have been extensively studied. Indeed, many algorithms have been published that arrive at best paths according to a given criterion. The problem is that the DCS is a network in which many of these results are not applicable. Since there are various types and priorities of users and additional constraints present, the conventional least cost path algorithms are not satisfactory. More sophisticated, general purpose constrained optimization techniques become intractable given the size of the search space.

We believe plan generation is an appropriate problem solving paradigm for the restoral problem. The key observation is that restoral is a set of individual tasks which are performed by a sequence of noncommutative actions. Changes are induced in the knowledge base describing the logical network structure as each step in the process of restoring a user is executed. The set of rules and conditions which constrain routing in a conventional approach serve as the rules directing either a forward or backward search through a state space of alternative network configurations.

Knowledge Required for System Control

A communications network consists of a large number and variety of equipments which form the essential resources which must be managed. To perform the various management functions, knowledge about the structure and the function of the communications network is needed. We have identified three general types of knowledge that are necessary for performance of the system control functions mentioned in the previous section. They are equipment configuration knowledge, equipment state knowledge, and communication paths knowledge. In the paragraphs which follow, we briefly discuss each of these categories and mention why each is needed to do fault isolation and restoral.

Equipment configuration knowledge is declarative knowledge about the physical configuration of the network. It includes pieces of information about specific pieces of equipment, where each piece of equipment is located, and to what it is connected. It also incorporates more general knowledge about specific equipment types, such as what status data is available and what operating characteristics are normal for the device. In the aggregate, equipment configuration knowledge is also knowledge about network topology. The connectivity of each individual piece of equipment to others, when taken as a whole, has implicit in it the entire network structure. Equipment configuration knowledge is clearly necessary in doing fault isolation. The network connectivity is needed in determining the ultimate source of a trouble because alarms propagate along links in general. The same knowledge is needed to determine plans for restoral because guaranteeing connectivity between two points ultimately requires knowledge of network connectivity to generate alternate paths.

Equipment state knowledge is somewhat similar to configuration knowledge but in fact is distinct. It is knowledge about the operating state of each piece of equipment. This may take the form of a simple statement such as "the radio transmitter at station ABC has a below average power output". It may also embody conclusions to be drawn from simple declarative information. For example, the statement concerning power output from a radio transmitter might be associated with the assertion "the radio receiver at the distant end of this link is likely to exhibit a sympathetic failure in received signal level". Such knowledge is clearly needed in sifting through a group of alarms indicating the operating status of the system in order to perform the fault isolation function. Similar information is used in generating plans for rerouting to determine alternative routes that are not likely to be acceptable.

Communication paths represent specific combinations of equipment which form a path between two or more users of a communications system. Thus communication path knowledge embodies knowledge at the level of a specific path (such as which equipments are utilized - radio, multiplexer, etc.). It also incorporates knowledge concerning appropriate actions to take in the event of interruption of a path. This knowledge is used primarily in the control of local node problem solving activity. The detection of an interrupted path in turn indicates a need for both fault isolation and planning for restoral.

In designing a knowledge base that incorporates the kinds of knowledge mentioned above, we felt that the structure of the systems that are the object of problem solving activity should be reflected as closely as possible. The reasons for this decision lie in the modes of problem solving to be performed using the knowledge present. As is evident from the discussion in the preceding section, network structure naturally guides the search for a solution in many situations.

The components of a physical network at the highest level are sites connected by links. Each site is named, has a physical location given by its latitude and longitude, is associated with its incident links, and has a local equipment configuration at the site. Each link is also named, has associated endpoints sites to which it is connected) and an associated media category. An example of a graphical representation for a portion of a network is shown in Figure 1.

Each major network component that is, each site and each link has behavioral characteristics that are determined by its attributes. For example, a link's characteristics are determined by its media [illegible]. This category, indicating

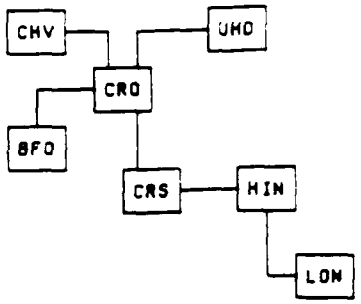


FIGURE 1

a link's type (i.e. microwave, cable, satellite, etc) and capacity, identifies a great deal of domain specific knowledge concerning the expected behavior of the link and the traffic it can and should be permitted to carry.

Each site, on the other hand, has an expected behavior determined largely by its local equipment configuration. Figure 2 gives an example of a "typical" site configuration. As can be seen in this example, a site contains a number of pieces of equipment: radios, multiplexers, switches, and crypto gear interconnected in a particular pattern. There may also be spare equipment ready to be put in service via remote control action. Sufficiently many different local configurations are possible (and even present in the current DCS) that it is not practical to categorize sites in the same broad manner that links can be categorized. Each specific type of radio, multiplexer, and switch has its own sets of alarms and status indicators. Thus each site's set of possible status signals may differ from that of others.

On top of the physical network, largely for the purposes of providing a structure for areas of control responsibility, is a designation of certain sites as subregional control facilities (SRCF). Each SRCF has a designated area of responsibility and each site, at any point in time, carries out control actions generated at its SRCF. This controlling superstructure must also be represented. The physical network serves as a carrier for many different logical networks, for example, packet switched data, dedicated data, and secure voice communications. Though the knowledge base design discussed in this paper does not model the existence of these subnetworks, we expect that they will be incorporated in future refinements of this design.

In the DCS, each user occupies a channel, several channels are multiplexed together to form a digroup, and digroups are multiplexed to form supergroups. The information concerning specific channels, digroups, supergroups, users, and traffic priority must be incorporated for the purposes of aiding circuit restoral. This knowledge is dynamic, changing relatively frequently in some cases while staying reasonably stable in others.

After considering many aspects of the problem, we have settled on a frame-based system to represent the knowledge associated with static components of the system. This allowed us to easily describe typical instances and situations, to provide default values and inheritance of properties, and to do procedural attachment of knowledge needed in controlling problem solving activity. Most of the task-specific knowledge is centered in the independent agents, each of which has access to the common knowledge base.

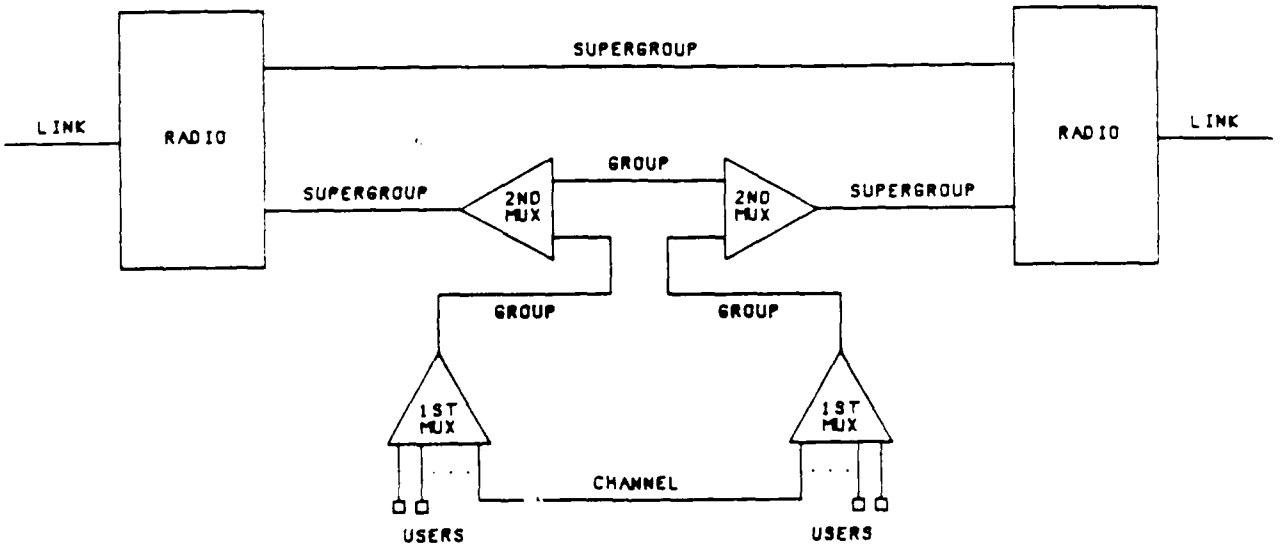


FIGURE 2

In Figure 3, we show an example of two sites connected by a link. The local site configuration as well as the link between sites is shown. Extrapolating from this example, we show the structural and logical relationships that are relevant to system control in Figure 4. This diagram is in the form of a semantic net like structure that explicitly exposes property inheritance features as well as physical and logical relationships.

Upon examining this structure, it is not difficult to identify several types of frames that should be incorporated. Frames for subregion, site and link instances are clearly necessary at a high level. At the level of physical equipment, structures capable of holding descriptions of radios, at least two types of multiplexers, and switches are required. Finally, frames for representing the groupings involved in the user, channel, digroup, and supergroup structure are needed. Examples of frame structures and their relevant slots are found in Figure 5. Knowledge concerning conclusions to be drawn and actions to be taken in the presence or absence of equipment specific alarms is contained in generic frames. These frames are linked to related instances by means of "isa" connections.

EXAMPLES

In this section we describe two highly simplified examples of problem solving activity involving fault isolation and restoral. In each case a number of assumptions have been made and only two or three domain specific rules are used in order to keep the example within the scope of this paper. The intent of these examples is to focus attention on the use of a shared knowledge base for two relatively independent problem solving agents. Both examples refer to the network structure given in Figures 2 and 3, as well as the knowledge base structure in Figures 4 and 5.

Fault Isolation - Example of Diagnosis

In this example we illustrate a knowledge-based diagnostic process which proceeds by first following a deductive reasoning path to a plausible set of hypotheses, each of which is then tested to identify the most likely candidate. The example does not address many of the detailed aspects of the problem such as coordination with distant problem solving agents, existence of conflicting sensory data, and incomplete or inaccurate knowledge of the network state and structure. Rather, this example is intended merely to illustrate the problem solving activity and the use of the proposed knowledge base.

The assumed failure is internal to multiplexer M2 at station LON and affects only the transmission of digroup G3 signals into supergroup S2. Since the fault is internal to M2, there may be no local alarms generated from station LON indicating the presence of this failure. The corresponding supergroup at station HIN is S4, and the corresponding digroup is G5. At station HIN the receive side of multiplexer M8 will detect the absence of a valid signal for digroup G5 and produce an alarm indicative of this signal loss. A similar alarm will be generated by the transmit side of M7, and by other multiplexers downstream (i.e. at stations CRS, CRO, etc.) until the digroup is finally demultiplexed into component channels. The existence of these alarms would trigger a performance assessment agent which determines the affected communications paths and modifies the equipment state knowledge base.

The fault isolation agent accepts sensory input of the following kind:

(station equipment alarm)

Based on the scenario described above, the following alarm reports would be observed:

(HIN M8 Rx input loss)
(HIN M7 Tx output loss)
(CRS M- Rx input loss)
etc.

In the frame for generic level-2-multiplexer, there are slots for each potential alarm. The fillers for these slots embody knowledge concerning the interpretation and impact of each specific alarm and even some combinations of alarms. In this case, the alarms at HIN on the multiplexers indicated cause formation of the following hypothesis:

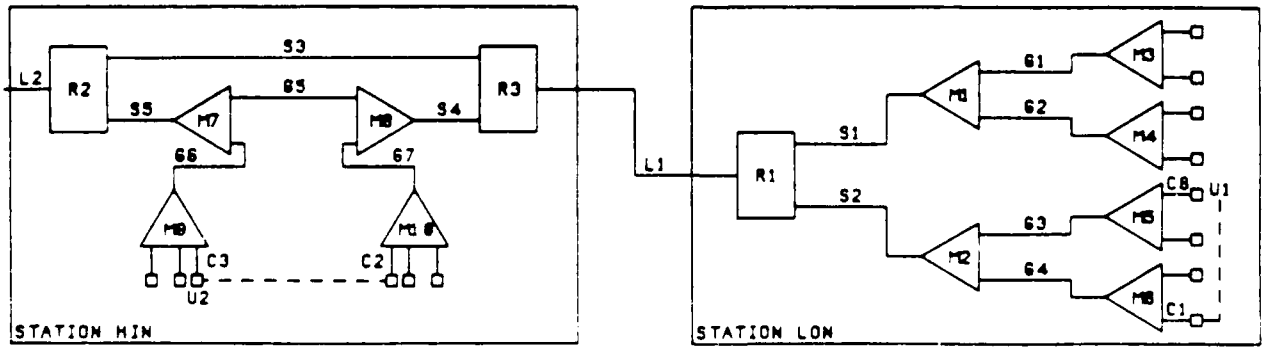


FIGURE 3

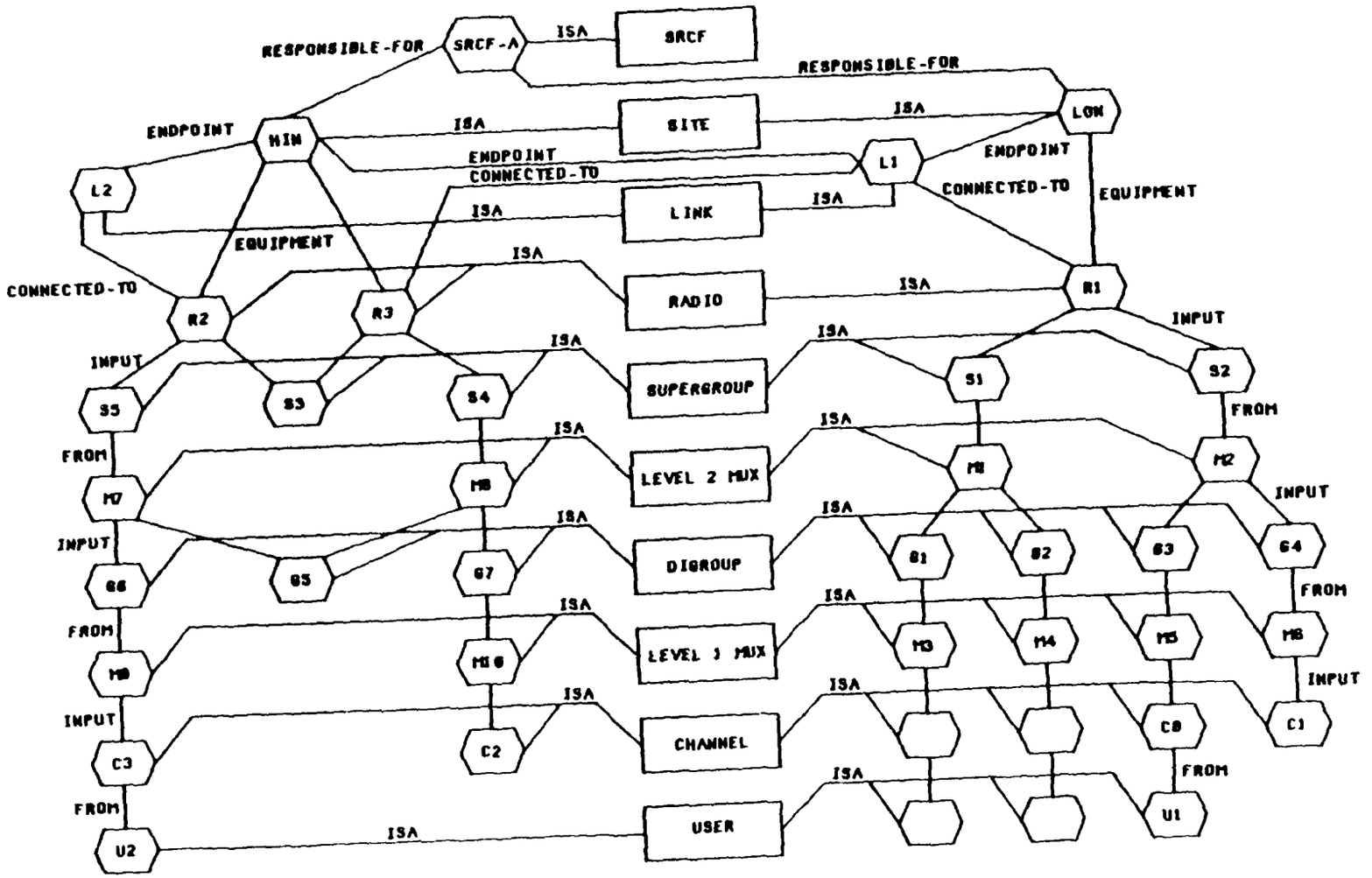

SRCF
SITE
LINK
RADIO
SUPERGROUP
LEVEL 2 MUX
DIGROUP
LEVEL 1 MUX
CHANNEL
USER
SRCF-A
MIN
LON
L1
L2
R1
R2
R3
S1
S2
S3
S4
S5
M1
M2
M7
M8
G1
G2
G3
G4
G5
G6
G7
M3
M4
M5
M6
M8
C1
C2
C3
C8
U1
U2
ISA
RESPONSIBLE-FOR
ENDPOINT
CONNECTED-TO
EQUIPMENT
INPUT
FROM

FIGURE 4

SITE INSTANCE
slots: name
location
isa
equipment
status
type
subregion-in
SRCF

LINK INSTANCE
slots: identifier
is
endpoints
status

SUBREGION INSTANCE
slots: name
isa
responsible-for
current-SRCF
backup-SRCF
adjacent-subregions

RADIC INSTANCE
slots: name
active-alarms
isa
connected-to
inputs
tx-on-line
rx-on-line

MULTIPLEXER INSTANCE
slots: name
isa
active-alarms
inputs
on-line

CHANNEL, DIGROUP, and SUPERGROUP INSTANCES
slots: name
isa
from
status

USER INSTANCE
slots: identifier
isa
priority

GENERIC FRAMES for:

SITE
RADIO
LEVEL-2-MULTIPLEXER
LEVEL-1-MULTIPLEXER
SWITCH
MICROWAVE-LINK
SATELLITE-LINK
TROPO-LINK
CABLE-LINK

CONTAIN SLOTS FOR:

equipment specific alarms

actions to be taken when certain alarms are active

consequences of alarm activation

FIGURE 5

- Multiplexer M8 at HIN has a partial failure

In the absence of an alarm of radio R3, another hypothesis is also formed. The trouble could have been caused further upstream than HIN, and multiplexer M2 at LON is the next piece of equipment that could have caused the problem. This is determined using physical connectivity (represented via the "input" slot values for the radio and multiplexer instances and the "connected-to" and "endpoints" slots of radios and links, respectively. As a result, there is an additional hypothesis, namely:

- Multiplexer M2 at LON has a partial failure.

Further refinement is not possible at this point without additional data, and thus a request for equipment tests would be made in order to resolve the ambiguity. The problem-solving paradigms involved are related to those used in the vehicular monitoring testbed developed at the University of Massachusetts [10,11].

Restoral - Example of Plan Generation

For this case, we assume the following initial operating state:

Initial State

1) User U1 at station LON is connected to a user at CRS via digroup G5 at intermediate station HIN.

2) All channels at LON are used, except C1 which is a spare and corresponds to C2 at HIN; likewise all channels at HIN are used, except C2.

3) All communication over digroup G5 is interrupted and the estimated time to repair is one hour.

A possible first step in generating a restoral plan for user U1 is to find an alternative digroup exiting LON. Given the rule which requires spare channels to be used in preference to pre-emption of lower priority users, we would expect the plan to suggest rerouting user UI to HIN via the spare channel. Generation of this step requires knowledge concerning equipment configuration, equipment state, and communication paths. This, of course, is not the desired final goal, but does move the network one step closer to that goal. In so doing, the equipment configuration knowledge base for stations LON and HIN have changed. The intermediate state is listed below:

Intermediate State

1) User U1 at station LON is connected to channel C2 at HIN.

2) All channels at HIN are used, but channel C3 is assigned to User U2 of lower priority than user U1.

3) User U2 is connected to a user at station CRS.

The next step is to find a channel from HIN to CRS for user U1. Given the rule which allows a higher priority user to pre-empt a lower priority user, the plan should suggest pre-emption of user U2 and making a new connection from channel C2 to channel C3 at HIN. The corresponding channel at CRS would be assigned to the designated end user for user U1. This requires the same sort of knowledge as the first step; however, changes have been induced in the knowledge base for both equipment configuration and communication paths. A new path now exists from LON to CRS via the patched channel connection at HIN. The final state includes the desired goal as listed below:

Final State

1) User U1 at station LON is connected to a user at CRS via digroups G7 and G6, and channels C2 and C3 at HIN.

2) User U2 at station HIN is currently pre-empted by a higher priority user.

It should be noted that the previous example does not embody the more complex rules and constraints actually used in routing. The use of spare channels and/or pre-emption, where necessary, can be modeled using conventional weighted graphs with least cost path algorithms. The more complex constraints are relevant to problems involving multiple restorals, avoidance of multiple pre-emptions for any single user, if possible, and requirements for spatially diverse routing paths among several users.

Concluding Remarks

In this paper we have described an application domain which requires concurrent activity in independent problem solving agents. Each of these agents requires access to a shared body of knowledge about the communications system, and each affects that knowledge base as a result of its activity. We have described a structure for a knowledge base which appears to be suitable.

This knowledge based structure is currently being implemented in ZETALISP on a Symbolics 3670. An object-oriented approach to programming is being used for its ability to describe abstract data types, which provide modularity and flexibility in a developing system. In addition, since many of the objects have homologous features, the implementation makes extensive use of flavors. For instance, objects such as radios, multiplexers, and switches can be classified as equipments, with generic operations such as updating active alarms, while other kinds of objects correspond to path components such as links, supergroups, channels, etc. which have similar properties. Also, the window system was found to be a convenient tool for providing the user interface and building network examples. Finally, we feel that the message passing semantics inherent in flavors captures the essence of communication in distributed problem solving.

References

[1] R. Davis and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence, Vol. 20 (Jan. 1983) 63-109.

[2] R.G. Smith and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving", IEEE Transactions on Systems, Man, and Cybernetics, Vol. C-129, No. 12 (Dec. 1980) 1104-1113.

[3] D. M. Schutzer, "Concept of Control of a Military Digital Communications Network", ICC 76 Conference Record, (June 1976), 34-7 - 34-13.

[4] E. H. Shortliffe, Computer-based Medical Consultations: MYCIN, American Elsevier, New York, 1976.

[5] R. O. Duda, P. E. Hart, N. J. Nilsson, and G. L. Sutherland, "Semantic Network Representations in Rule Based Systems" in D. A. Waterman and F. Hayes-Roth (Eds.), Pattern Directed Inference Systems, Academic Press, New York, 1978.

[6] R. Davis, H. Schrobe, W. Hamscher, K. Weickert, M. Shirley, and S. Polit, "Diagnosis Based on Description of Structure and Function", Proceedings of AAAI-82, August 1982.

[7] M. S. Fox, S. Lowenfeld, and P. Kleinosky, "Techniques for Sensor-based Diagnosis", Proceedings of IJCAI-8, August 1983.

[8] T. Finin, J. McAdams, and P. Kleinosky, "FOREST - An Expert System for Automatic Test Equipment", Proc. First Conference on Artificial Intelligence Applications, (December 1984), 350-356.

[9] F. Pipitone, "An Expert System for Electronics Troubleshooting Based on Function and Connectivity", Proc. First Conference on Artificial Intelligence Applications, (December 1984), 133-138.

[10] D. Corkill and V. Lesser, "The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network", Proc. of IJCAI-8, August 1983.

[11] D. Corkill, "A Framework for Organizational Self-Design in Distributed Problem Solving Networks", PhD Thesis, Univ. of Mass., Amherst, Mass., 1983.

4 Planner System for the Application of Indications and Warning

Report Submitted by:
Sergei Nirenburg
Colgate University

Table of Contents

1. Introduction 278
2. Strategy 278
3. Technical Content 279
3.1. POPLAR 1.3 279
3.1.1. Introduction 279
3.1.2. Setting the Stage 279
3.1.3. The Conceptual Architecture of POPLAR 1.3 280
3.1.4. The Implementation 283
3.1.5. The System Architecture of POPLAR 1.3 283
3.1.6. The Algorithms 290
3.2. POPLAR 2.0 294
3.2.1. Introduction 294
3.2.2. Comparing the Functionality of POPLAR 1.3 and POPLAR 2.0 294
3.2.3. Improvements in POPLAR 2.0 297
3.2.4. Plans in POPLAR 2.0 297
3.2.5. The Algorithms 299
3.2.6. Implementation 300
3.2.7. An Example 301
4. Background and Related Work 303
5. Status and Future Development 306
5.1. General 306
5.2. Specific Plans: POPLAR 3.0 307
Bibliography 308
Appendix 1. Representation of OBJECTS in POPLAR 1.3 311
Appendix 2. Examples of PLAN representation in POPLAR 1.3 315
Appendix 3. Examples of POPLAR 1.3 rating functions 319
Appendix 4. HISTORY in POPLAR 1.3 321
Appendix 5. BLACKBOARDS in POPLAR 1.3 321
Appendix 6. Examples of PLAN representation in POPLAR 2.0 323
Appendix 7. Providing Intelligent Assistance in Distributed Office Environments 327
Appendix 8. POPLAR: A Testbed for Cognitive Modeling 345

1. Introduction.

The Colgate University project in the framework of the North-East Artificial Intelligence Consortium is devoted to the design of a planner system for the application of I & W (Indications and Warning). The specification of the task as above has evolved in the course of this past year from the early direction of intelligent database management toward the emphasis on the problem-solving activity. The task of the project is two-pronged:

A. Design of a system that will
   1. obtain as input messages concerning events in a model of a real-life subworld;
   2. 'understand' these events by detecting what *plans* they are parts of and, whenever applicable, what *goals* are pursued by the instigators of these events;
   3. produce (suggestions for possible) plans of action necessary in connection with the situation in the world.

B. Implementation of this system for the world of I & W (the Indications and Warning application).

This general task includes a large number of subtasks, many of which require significant research effort. We have concentrated on designing the mechanisms and knowledge bases for the problems of plan recognition (a part of 'understanding' in 2. above) and plan production (in the framework of 3. above). We excluded from our consideration the problems of perception (speech, graphical, or visual); the problem of understanding natural language inputs (that is, understanding the contents of messages), as well as actual performance of plans suggested by our system.

The conceptual background of this effort is described in some detail in Section 4. The historical background of the project and the way in which it merges with other efforts in the consortium is briefly discussed here. Cooperation with other research teams within the consortium, especially with the University of Massachusetts project led by Victor Lesser and Bruce Croft, led to the state of affairs where plan understanding and plan production has become the main thrust of the research effort at Colgate. It was decided, in consultations, with the project monitors at RADC, that the natural language aspects of the task will be postponed till the next stage of the research.

2. Strategy.

We have taken a concentrical approach to the task of designing and implementing the planning system. In other words, we decided to produce an implementation for every design version of our system (called, for historical reasons, POPLAR). As our study of the problem of knowledge-based automatic planning progresses, newer versions of the system will appear.

In what follows we describe in succession the design peculiarities and the implementation characteristics of the two versions of our system (POPLAR 1.3 and POPLAR 2.0) that were developed over this past year. Next we set the goals for the

implementation of the next version of the system (POPLAR 3.0). This last version introduces very substantial changes in the overall design, and the next year of the project will be devoted to their implementation.

3. Technical Content.

3.1. POPLAR 1.3.

3.1.1. Introduction.

This paper presents an overview of a cognitive modeling system centered around a personality-oriented planner, and then describes in detail the types of knowledge it uses to make control decisions. POPLAR is a model of an intelligent actor capable of planning sequences of control and domain actions in a simulated world that exists independently of the planner. The world is a simplification of the 'Dungeon' computer game environment. The actor makes control decisions on the basis of situational knowledge as well as its personality characteristics (character traits, physical and mental states) and its beliefs about personality of other cognitive entities in the world. POPLAR is a step toward an AI system whose behavior is psychologically justified and can provide the basis for an experimental testbed in cognitive modeling.

3.1.2. Setting the Stage.

The POPLAR planner is a component in a model of an intelligent actor. It is an approximation of the human actor in that:

i) like humans, it possesses multiple goals with associated plans;

ii) like in humans, its control decisions depend upon multiple sources of information, e.g. input from the 'objective' world, its permanent character traits, its temporary physical and mental states, and past experience;

iii) like humans, it is immersed into an 'objective' world, changes in which can be introduced not only by the actor, but also by events beyond the actor's control, making it necessary to deal with non-monotonicity.

We believe that the essence of an intelligent actor's cognitive activity is best described in terms of the following loop:

1) perceive input stimuli (sensory, proprioceptive or mental);

2) generate goals connected with these stimuli;

3) schedule the most important goal instance for the given period of time: the one to which the actor's cognitive resources are allocated;

4) choose (occasionally, create) and

5) execute plans to achieve this goal, including performance of physical, verbal or mental actions that are components of these plans. Executions of the loop provide continuous change and stimulation at several levels. Physical actions introduce changes in the objective world. Verbal actions can provide sensory input for other

intelligent actors in the world. Mental actions introduce changes in the world of the actor himself (his event memory and beliefs). So, the actions by the actor and other actors in the objective world change this world, and therefore, provide new inputs for the system.

POPLAR 1.3 offers a solution to above loop components 2), 3), the non-creative part of 4), and the mental action part of 5). The visual perception portion of 1) and the physical actions of 5) are **simulated** through interaction with the human user of POPLAR.

In the current implementation there is no natural language capability (i.e. the verbal behavior of 1) and 5) are not addressed). Nor do we tackle in any complete and principled manner the extremely complex problem of learning (one facet of which is the creative part of 4).

The central cognitive and architectural points that distinguish POPLAR 1.3 are, in addition to i) - iii) above, as follows:

A. The choice of the type(s) of knowledge for scheduling (cf. 3 above) and selecting (cf. 4 above) activities. We proceed from the assumption that in a non-trivial world these operations should be based on a psychologically justified model of human cognitive behavior. This property makes POPLAR 1.3 personality-oriented, i.e. provision is made in the present model for introducing personality factors that influence goal generation and plan selection.

B. Decisions concerning the organization of metaknowledge that monitors and directs the cognitive processes of goal generation and plan selection. POPLAR 1.3 represents such metaknowledge in the same framework as the domain plans (top-level, intermediate and primitive). This allows them to be processed by the same reasoning mechanism.

3.1.3. The Conceptual Architecture of POPLAR 1.3.

The conceptual architecture of POPLAR 1.3, as presented in Figure 1, consists of the following modules:

1) the objective **world**, information from which and from
2) the **regulatory system** of the actor (cf. Norman, 1981), where the non-cognitive knowledge about the actor's character and physical and mental states is stored, is obtained by
3) the **sensor**, which processes this input and produces, in the **short-term memory** (STM) of an actor,
4) the **snapshot**, in which the objects currently perceived by the actor are stored, with their parameters, to be scanned by
5) the **goal generator** component of the reasoning mechanism (the **cognitive** module) which produces

6) the list of **candidate goals**, that contains all the goal instances that the actor has at a certain time, including the ones added after the new input was processed. In making its decisions, the **goal generator** uses the data stored in

7) the actor's **long-term memory** (LTM), which contains knowledge about

   a) the **beliefs** the actor has about

   -- **objects** in the objective world, including self-beliefs

   -- actor's **goals**

   -- domain-specific and metalevel **processes** (stored as **plans**)

   b) the acquired **values** the actor has about these beliefs: what is more important, when and why, etc.

   c) the **event** memory that embodies past experience

8) The **scheduler** component of the reasoning mechanism chooses (schedules) a goal instance in the list of candidates and selects the appropriate plan for its achievement. The **executor** component of the reasoning mechanism then attempts to execute the plan. Lower-level primitive plans are, in fact, actions that are performed by

9) the **output** module; these actions can introduce changes into the world, into the list of candidate goals and the long-term memory.

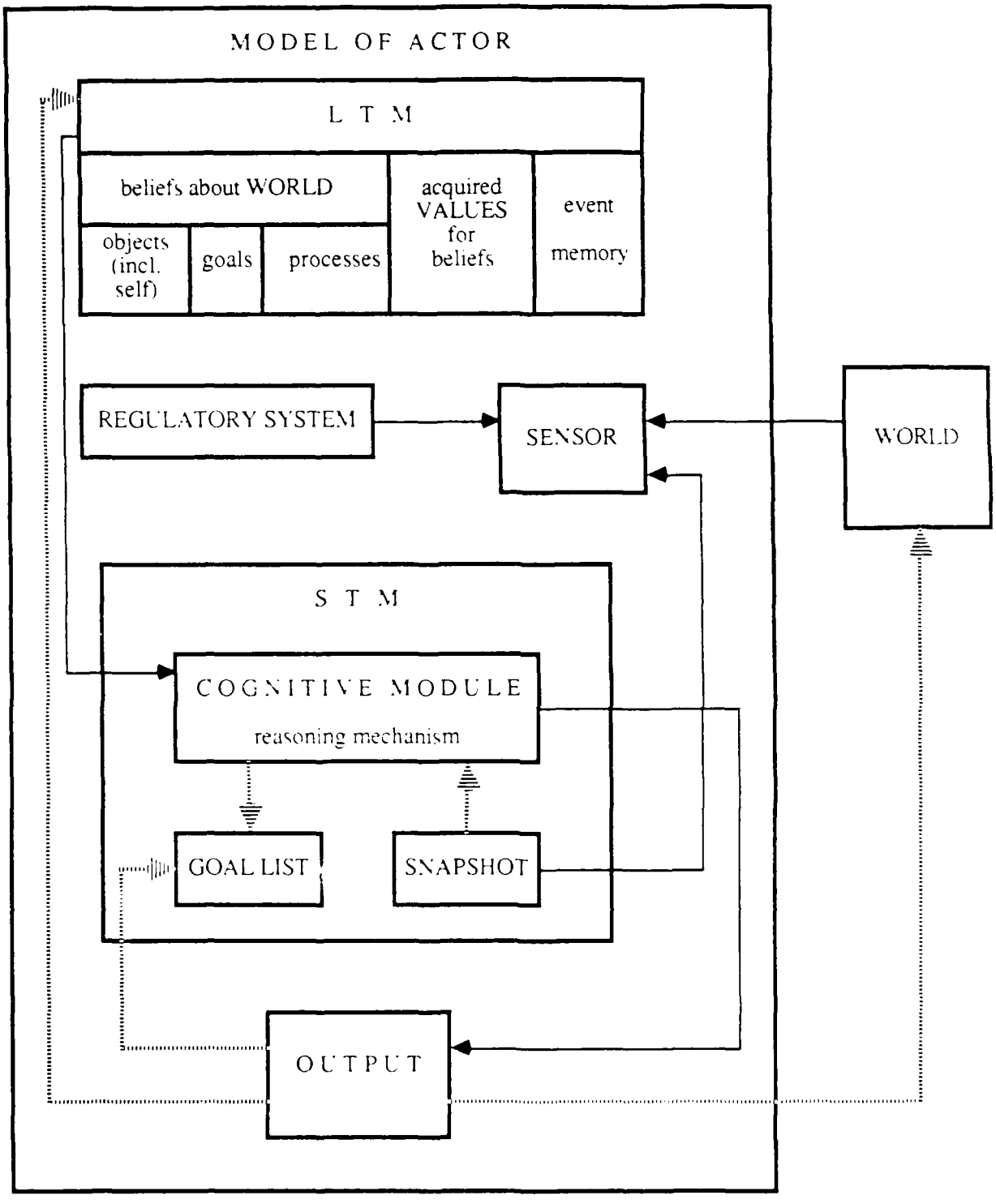


Figure 1. The conceptual architecture of POPLAR.

3.1.4. The Implementation.

POPLAR 1.3 is an implementation of the above conceptual schema in a concrete application domain. It has been implemented in PEARL (Deering et al., 1981) which runs in Franz Lisp under Unix 4.2.

The world in which POPLAR 1.3 is immersed is reminiscent of that of the well-known 'Adventure' or 'Dungeon' games. We represent a cave in which POPLAR's actor can find and react to enemies, treasures, tools, weapons, food and other objects. It is important to understand, however, that POPLAR 1.3 is **not** a game-playing system. We are in the process of applying the system in a different domain (the office world).

At present POPLAR's actor is supplied with three basic goals:

1) 'Don't get killed', dubbed Preserve-Self-1 or PS1

2) 'Don't die of hunger, thirst or fatigue', Preserve-Self-2 or PS2

3) 'Collect as much treasure as possible', Get-Treasure or GTR.

In POPLAR 1.3 the system is making the decisions about what to do next, while it is the responsibility of the user to provide it with input and means for verification of success of actions. The user, therefore, provides the testing ground for the system's empirical experience in the world.

With this caveat in mind, let us see how POPLAR 1.3 is organized to allow its actor to 'act' in this environment.

3.1.5. The System Architecture of POPLAR 1.3.

POPLAR 1.3's system architecture (Figure 2) represents the conceptual architecture of Figure 1 with implementation restrictions superimposed.

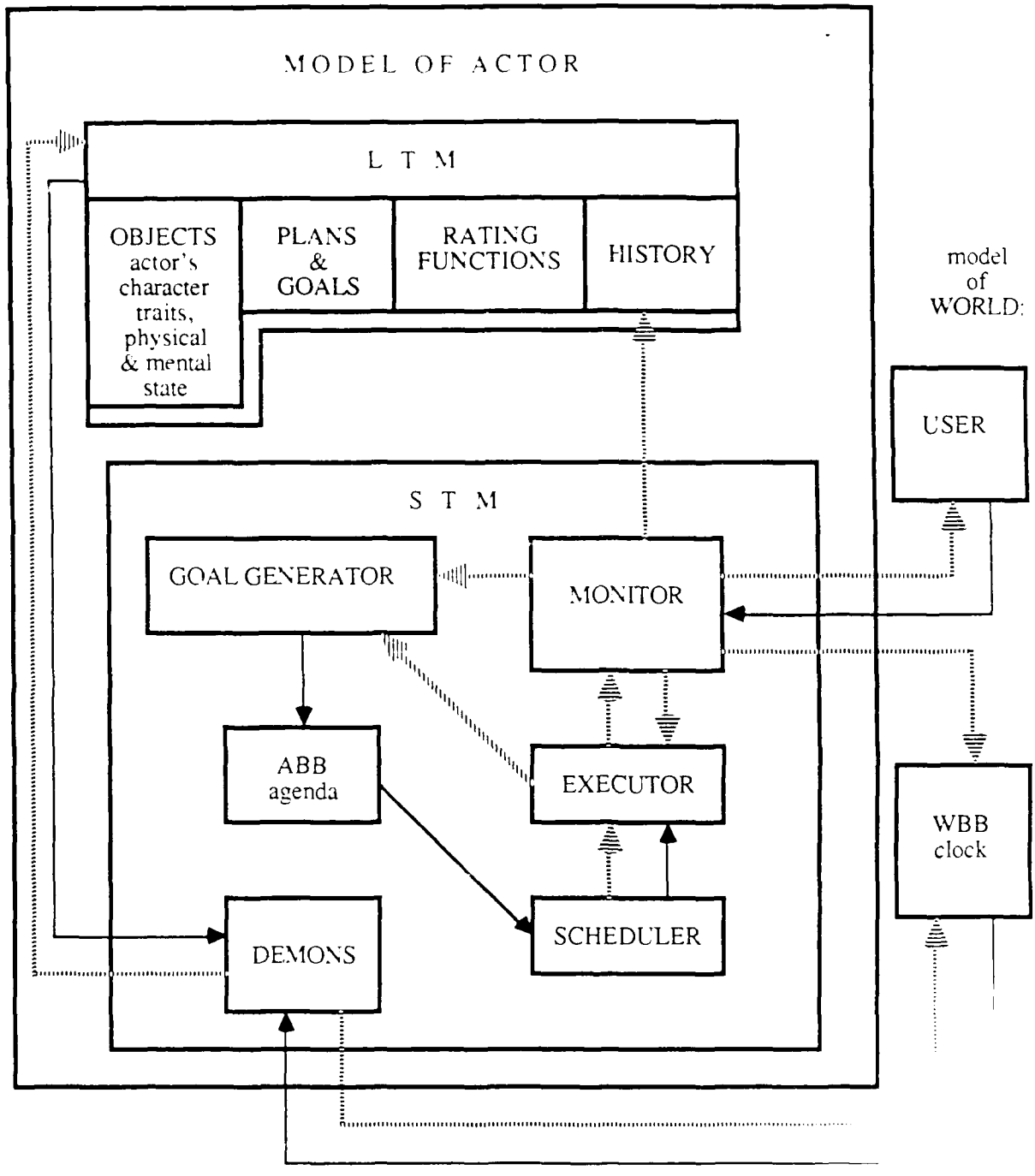
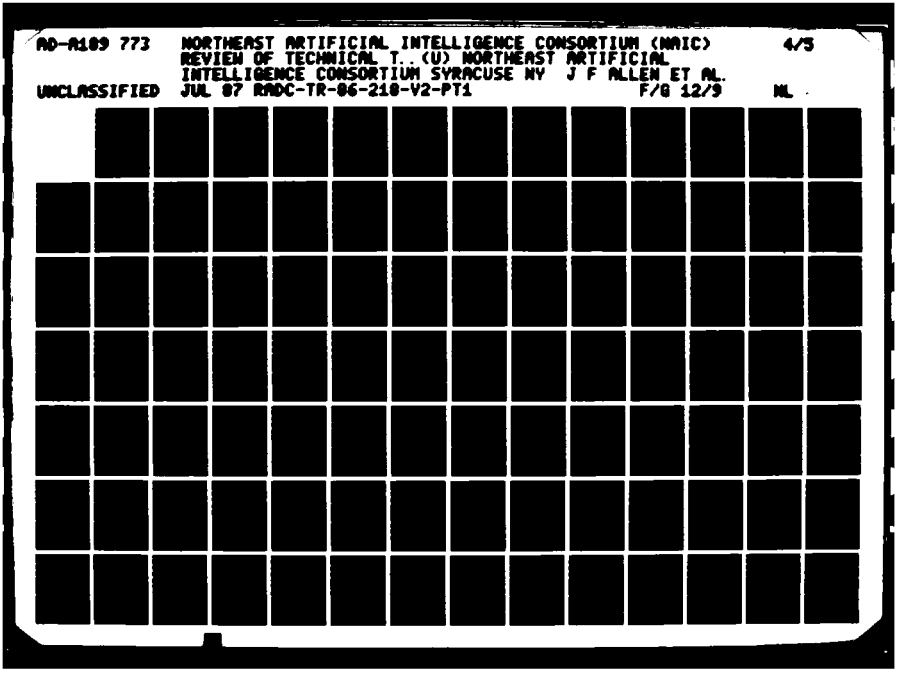
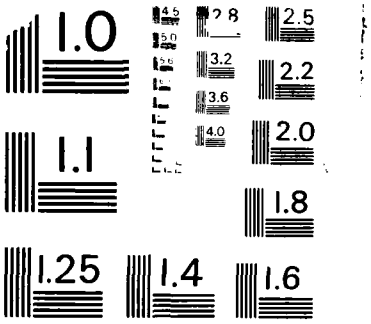


Figure 2. The system architecture of POPLAR 1.3


AD-A189 773 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC) 4/3
REVIEW OF TECHNICAL T.. (U) NORTHEAST ARTIFICIAL
INTELLIGENCE CONSORTIUM SYRACUSE NY J F ALLEN ET AL.
UNCLASSIFIED JUL 87 RADC-TR-86-218-V2-PT1 F/G 12/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963

In POPLAR 1.3 the role of the objective world including the provision of its rules, 'the laws of nature', is assumed by the human user experimenter. The user also interactively introduces and removes objects in the cave and modifies their parameters. (In future versions we intend to implement ongoing changes in the objective world generated by the operation of if-added demons on a World Blackboard (cf. 3.1.5.2.1).

The user also either permits or forbids certain primitive operations to simulate the actor's pragmatic experience. For example, the user might forbid the actor to pick up an object that is 'too heavy' but previously believed by the actor to be manipulable. This natural state of affairs underscores the difference between the objective world and the world of POPLAR's actor and his beliefs. It is also a means of modeling mistakes (a necessary first step in trying to learn how to recover from them).

The sensor and the output block are simulated in POPLAR 1.3's monitor (though mental actions are performed by demons (see below).

When the user decides to add an object to the current world, it does it by listing it on the **world blackboard** (WBB), the data structure interfacing the objective world and the world of POPLAR's actor. WBB also contains a clock which guides all temporally spread processing

The STM of POPLAR's actor has the reasoning mechanism (the monitor and the executor with their associated bookkeeping functions, demons) permanently connected with it*. STM contains one-instance metaplans: the goal generator and the scheduler. STM also includes the **actor blackboard** (ABB), which contains slots relating to the current state of POPLAR actor's activities, including notably the **agenda** of activated goal instances.

POPLAR actor's LTM contains his objects, plans, rating functions and history. Cf. a detailed discussion in 3.1.5.1.

POPLAR actor's knowledge about his own regulatory system and that of others is linked in the implementation with the representation of these objects in LTM. In addition to knowledge about objects, LTM contains knowledge about plans, history of processing and proper scheduling and selection.

Let us discuss the components of POPLAR 1.3 in greater detail.

3.1.5.1. LTM.

3.1.5.1.1. Objects.

Several typical object frames and the semantics of their slots are described in Appendix 1. The choice of character traits is at present empirical. However, in parallel to implementing POPLAR 1.3, we have been conducting extensive psychological experiments seeking to establish the set of 'primitive' personality characteristics and their

* The monitor, the executor and the bookkeeping functions stand out among the components of STM in that they are not 'conscious' functions the actor performs them 'instinctively' while of other elements of STM the actor is consciously aware

mapping into more complex notions that are used by intelligent actors in personality-based decision-making. A separate set of experiments will determine the primitives for specifying mental states of the actor.

3.1.5.1.2. Plans.

POPLAR's knowledge about the dynamics both in the objective world an in the actor world is represented as a set of declarative structures called plans.

Plans in POPLAR 1.3 are classified into several groups (cf. Figure 3).

	TOP-LEVEL	INTERMEDIATE	PRIMITIVE
DOMAIN PLANS	PS1 PS2 GTR	FIGHT, EAT, GET, etc.	move, take, find, etc.
METAPLANS	GG as		as, gg-input, etc.

Figure 3. Classification of plans in POPLAR.

First, there are domain plans that describe actions in the world and metaplans that describe the processes that manipulate other plans. These include such plans as the goal-generator (gg), the plan-selector, the agenda-scheduler (as), etc. Second, there are top-level plans whose instances appear in POPLAR's agenda as representatives of the three main goals; and primitive plans that are no further decomposable into sequences of actions and provide the proper framework (of preconditions, effects, etc.) for their main action.

The plans that are neither top-level nor primitive are called intermediate. Intermediate plans are never scheduled other than in the process of expanding a top-level plan. There are no intermediate metaplans. Also, all of the metaplans are primitive (decomposable), and two of them, at the same time, top-level.

To illustrate the above discussion, consider, for instance the top-level plan of dealing with enemies, such as, in the POPLAR 1.3 objective world, snakes, crocodiles or trolls. The actor can have a number of (intermediate plan) possibilities: to fight, to flee, to hide, to wait and see what happens, etc. All of the above are decomposed into strings of lower level plans (such as get, take, find, etc.), and the process of decomposition continues until all the final decompositions contain only primitive plans (such as, for instance, move or take).

Plans in POPLAR 1.3 are represented in a modified version of the language EDL (Bates et al., 1981; cf. also Croft & Lefkowitz, 1984). The frame for a plan contains the following slots (clauses):

ID	the name of the plan
TOP-LEVEL-FLAG	is this plan top-level?
IS	contains the temporal and causal expansion of the plan
COND	used to pass parameters ('propagate constraints') to lower-level plans upward propagation will be added for the plan recognition task
WITH	specifies the parameters with which the current plan will be processed
CONTROL	contains predicates to choose whether to execute optional steps in the plan this slot has the form of an a-list (<(Control# <s-expr>)>*
PRECONDITIONS	predicates that allow the processing of the current plan to start, differ in principle from CONTROL predicates by being independent of the current context of plan processing
STATUS	one of 'on-agenda' 'executed', 'succeeded', or 'failed', used for communications with the reasoning mechanism
ACTION-FOR-PRIMITIVE	if plan is domain primitive permission is requested for its completion and the main action is performed (the rest being 'effects')
TIME	number of time cycles the plan takes (only for primitives) -- either integer or s-expression that evaluates to integer
RATING-FUNCTION	scheduling knowledge, see below
EFFECTS	auxiliary (including bookkeeping) modifications accompanying the success of the plan

Figure 1 contains a grammar of the plans implemented in POPLAR 1.3, and Appendix 2 contains annotated examples of POPLAR 1.3 plans.

I ::= PS1 | PS2 | GTR | GG ('Goal-Generator') | as ('Agenda-Scheduler')
PS1 ::= FIGHT | HIDE | WS ('Wait-and-See')
PS2 ::= EAT | DRINK | SLEEP
GTR ::= {FIGHT | find} GET
GG ::= gg-input | gg-objects-perceived | gg-physical-states-perceived
FIGHT ::= find {find GET} move attack
HIDE ::= find move
WS ::= do-nothing
EAT ::= find {find GET} ingest
DRINK ::= find {find GET} ingest
SLEEP ::= find do-nothing
GET ::= move take

Vertical bars separate disjoined elements; in practice,
the 'or'-ed plans are chosen on the basis of their ratings
through the application of a special metaplan we call the
Plan-Selector, not shown in the grammar;

curly brackets enclose optional plans; the decision
whether to execute the optional plan(s) is made on the basis
of control functions that are stored in the parent plan and
govern the parsing of its IS slot;

plans shown in lower case are primitive.

Figure 4. A grammar of plans in POPLAR 1.3.

3.1.5.1.3. The Rating Functions.

The knowledge that POPLAR's actor has about the relative importance of a top-level goal instance and the relative merits of one plan of action aimed at achieving a goal over another is embodied in the **rating functions**. In the current implementation rating functions are associated with every plan that can serve as parameters in the plan-selector and the agenda-scheduler.

The rating functions calculate a numerical value for a plan, a rating, in all situations where a choice among plans that can be pursued is possible. They draw upon:

a) knowledge of the objects involved in an objective world situation;

b) the character traits, mental and physical states of the actor;

c) the actor's beliefs about the character and current physical/mental state of any other cognitive entity participating in the situation;

d) the actor's event memory, the history of past processing.

Thus, if two actors, Actor1 and Actor2 find themselves in an identical threatening situation (e.g. a snake), but one of them is more courageous (a character trait) and or is in general not very fearful of snakes (a situational characteristic), the actors may

respond to the situation by choosing different plans (e.g. Flee for Actor1 and Fight for Actor2) or even altogether different goals (while Actor1 is likely to choose 'Preserve-Self' against the snake -- because high levels of attention to threats can be expected from actors with low courage values; Actor2 may choose, say, an instance of 'Get-Treasure', because the snake is not serious enough a threat).

The construction of rating functions is an empirical process of gradual refinement. Even without changing the knowledge used by the rating functions one can always manipulate parameters of a function to calibrate its results.

One of the objectives of the psychological experimentation conducted in parallel with this project (cf. Section 4) is to better understand the nature and parameters of the rating functions.

Examples of rating functions are presented in Appendix 3.

3.1.5.1.4. History

This part of the actor's LTM contains his memory of past processing. In principle, history can have a very rich structure and be used in a wide variety of ways. Special demon-type functions can be defined, for example, to introduce modifications into the actor's beliefs about objects and processes in the real world based on certain patterns in the event memory[4]. This is one more location in POPLAR 1.3's architecture where a measure of learning can and is planned eventually to be introduced.

At present the history contains only two types of data: a) the record of all the recursive calls to the executor in the form of paths that the processing took in the grammar of plans and b) a list of the objects (physical or mental) found by all instances of the Find plan; this knowledge is used to retrieve the status and the results of various plan instances. A typical instance of history is presented in Appendix 4.

3.1.5.2. Actor/World Interfaces.

As mentioned above, in the current implementation of POPLAR 1.3 there are two blackboards that facilitate links between the world and the actor.

3.1.5.2.1. The World Blackboard.

WBB is used for introducing new sensory input and managing temporal relations in the system. POPLAR 1.3 has time-triggered demons that automatically update the values of the actor's physical and mental state based on the amount of time he engages in a certain activity.

---

[4] An example: Suppose that in an internalized plan for fighting crocodiles 'stick' is listed as the best weapon. Then during one invocation of the plan Fight (Actor Crocodile Weapon) no stick could be found so that Actor had to use a gun. It appeared that both the results were better and the fatigue increase was smaller. After this plan execution was written into the history, a comparison is made (by the above demons) and the old belief about the stick being the best weapon is changed.

In their simplest manifestations, the time-related modifications deal with increasing the actor's hunger, thirst and fatigue values at predetermined independent rates. When the value of any of the above parameters becomes greater than a predefined threshold, a message to this effect automatically registers in ABB's 'states-perceived' slot, as a result of which at the next pass of the monitor an instance of Preserve-Self-2 goal will be activated, and the corresponding top-level plan will appear on the agenda.

Temporal knowledge is also used to implement a simple model of attention. A detailed discussion of this mechanism will be deferred till Section 3.1.6.3.

3.1.5.2.2. The Actor's Blackboard.

ABB contains information about

a) the list ('objects-perceived') of object instances that the actor has perceived in the current environment;

b) the list ('states-perceived') of all physical states currently perceived that warrant the attention of the goal generator (e.g. the level of hunger above a threshold);

c) the agenda of all top-level plans (the representatives of the main goals) vying for the attention of the cognitive processor of the actor at any given time;

d) the stack ('current-path') of plans currently being executed (from a top-level plan to a primitive).

In future implementations, specifically when plan recognition will be added to the repertoire of POPLAR and the number of actors inhabiting its world will be allowed to be greater than one, the number of ABBs in the system may grow to as many as the square of the number of actors. This is because every actor stores his beliefs about other actors' activities in instances of ABB attached to his representation of these other actors. Therefore, each actor theoretically can be aware of all the other actors and contain an ABB for each, including himself.

A typical example of ABB and WBB contents is presented in Appendix 5.

3.1.6. The Algorithms.

3.1.6.1. The Monitor.

The top-level control function of POPLAR 1.3, the monitor, is an infinite loop (our actors do not die -- only if killed by enemies!) which performs the following tasks:

a) it maintains contact with the user (to obtain new input);

b) it starts the executor loop that consists of i) processing new input; ii) scheduling an action; and iii) executing this action

c) it displays selected situations in the world with the help of a (rather simple) graphic interface.

3.1.6.2. The Executor.

The main bulk of POPLAR 1.3 processing is performed by the executor. To understand how POPLAR 1.3 works it is sufficient to trace a cycle of its activities.

The executor is called many times during one monitor cycle. First, it processes the goal generating plans using the information obtained by the monitor from the objective world as well as that from the actor's regulatory system. As a result of this stage, the agenda of competing top level plan instances is updated. Second, it executes the agenda scheduler plan select the best candidate plan. Finally, it executes the chosen top-level plan (this involves a number of recursive calls to the executor). When eventually the execution ends, the result of current processing (success or failure) is reported, and a new cycle of the monitor begins.

Omitting a few overly technical details, we can describe the activities of the executor generally as follows:

a) obtain a plan to process; if it is not a plan **instance** (the agenda holds only plan instances, e.g. 'GTR19'; whereas **is** clauses of plans are formulated in terms of plan **types**, e.g. 'Find'), create a new instance of this plan;

b) check the plan's **preconditions** clause; if preconditions do not hold, report failure and its reason and exit; otherwise,

c) expand the plan by considering its **is** clause: call the **is** clause parser;

   c') if the **is** clause is 'primitive', then **action-for-primitive** is performed (most often this is a request to the 'laws of nature', the user, to allow an update in the objective world, e.g. a move by the actor; if the permission is given the processing proceeds as specified in e') below; if the action is not allowed the processing proceeds as in e''). (Let us repeat that the semantics of this situation is that the actor's beliefs about the objects and/or plans and/or values are somewhere wrong, as a result of which some indication of imminent failure must be given to prevent the 'automatic' success of most planners in situations where the internalized preconditions of a plan hold.)

   c'') if the IS clause is not 'primitive' the parser has to make specific control decisions: i) whether to execute an optional subpath in the IS clause; ii) which of any possible number of disjoined subplans to choose for fulfilling the current plan. (The ability to choose one of a number of 'shuffled' subplans (those that can be fulfilled in any temporal order) will be added to POPLAR in near future.) The knowledge about whether to execute an optional subpath is encoded in the **control** slot of the plan whose **is** slot is parsed. The knowledge selecting one of disjoined subplans is contained in the plan-selector metaplan and the **rating function** slot of the current plan. Once it becomes clear what member of the **is** clause should be processed first, the executor

d) calls itself recursively with this plan; this event is recorded on ABB, specifically, in a data structure called **current-path**; the old content of **current-path** is added to **history**.

e′) *if an* **is** *clause is processed to its end (cf. the special case of 'primitive' in c′)* above), the **status** slot of the plan is set to 'succeeded' and the **effects** clause is evaluated;

e″) if for some reason the **is** clause cannot be processed to its end, the **status** slot is set to 'failed' and

f) this information is communicated to the parent plan; the current plan is discarded from the **current-path** stack, and the processing of the **is** clause of the parent resumes. When, eventually, the outcome of the top-level (and bottom-of-stack in **current-path**) plan becomes known, then

g′) if it succeeded, then the **effects** clause is evaluated and the corresponding top-level plan instance is removed from the agenda (and added to **history**);

g″) but if it failed, then, assuming that the need that had spawned this goal has not been satisfied, the executor creates a new instance of the same top-level plan and adds it to the agenda instead of the failed one (which goes to **history**).

h) a new cycle of the monitor starts.

3.1.6.3. Modeling Attention.

The previous section described the normal flow of control in a monitor cycle. In real life, however, an actor can hardly have the luxury of being able to finish the processing of a top-level plan without taking in new information about the objective world. In future implementations of POPLAR the temporal relations among plans will be elaborated to include the many possibilities of concurrent processing (cf. Allen, 1983a, for the description of a model of time that can be adapted for use in our model; cf. also McCue & Lesser, 1983) for a temporal logic in the POISE system).

POPLAR 1.3 reacts to this problem as follows. When a top-level domain plan is chosen from the agenda and passed over to the executor, its rating is used for calculating the number of time cycles this plan will be allowed to execute without being interrupted. The more 'important' the plan (i.e., the higher its rating) the longer it is allowed to execute uninterrupted. This current programming device is a rough simulation of the actor's concentration or attention to the task. Intuitively, the more immersed one is into a task, the less one would be inclined to be distracted by new sensory inputs. It is obvious that character traits and physical mental states affect the ability to concentrate.

When an interrupt occurs, the entire **current-path** is suspended; the instance of the top-level plan is deleted from the agenda and another instance is created and added to it (the new instance reflects the knowledge of the stage at which the processing was suspended; **history** is used for this purpose). Then the monitor starts a new cycle.

3.1.6.4. An Example.

Suppose we want to test the performance of POPLAR 1.3 in the following situation of the world. We want to put the actor in a cave with a rock, a snake and an apple and to set its hunger well above the detecting threshold.

POPLAR 1.3 acts as follows:

a) asks the user whether he wants to remove certain objects from the world: we do not, so we answer in the negative;

b) asks the user whether he wants to change any of the properties of the objects already present in the world; this is the time to input the (high) value of actor's hunger;

c) asks whether the user wants to add new objects to the world; we do; since our perception module is simulated, we submit prefabricated instances of objects to POPLAR 1.3; we write: (rock1 snake1 apple1).

d) adds the above object instances to ABB.objects-perceived. Since snakes spawn the need for protection (by virtue of their being descendants of 'creature'), the goal Preserve-Self-1 is activated (by the gg-input plan) and an instance of its corresponding top-level plan, PS10, is added to ABB.agenda (which already contain the unique instance of the Agenda-Scheduler plan that resides there permanently); appropriate messages are issued by POPLAR 1.3;

e) detects, through gg-states-perceived, the actor's hunger; 'hunger' is added to ABB.states-perceived and an instance, PS20, of the top-level plan of the Preserve-Self-2 goal is added to ABB.agenda; appropriate messages are issued;

f) since no objects had been present in the world before, and, therefore, no changes to their properties could be introduced, gg-objects-perceived will not be needed in this case, a message to which effect will be issued;

g) at this point ABB.agenda is (agenda-scheduler PS10 PS20); the monitor calls the executor with the scheduler plan, as a result of which the two domain plans receive ratings. Suppose now that PS20's rating is higher (because the actor is very hungry and at the same time not too afraid of snakes); this being the goal choice,

h) the scheduler is called with PS20(Actor hunger); checks its preconditions (empty!) and expands its **is** clause; the plan-selector, using the rating functions in the plans Eat, Drink and Sleep, decides to select Eat; an instance of Eat, Eat0(Actor) is created and pushed onto **current-path**

i) Eat0's preconditions are checked (empty!), and its own **is** clause is expanded; this means creating a new instance of Find, Find0(Actor food Actor.inventory), -- that is, first the actor wants to check whether he is carrying some food;

j) the control1 predicate chooses whether to execute the optional Find and Get plans; the predicate essentially returns 'true' if the previous Find failed; the optional subpath corresponds intuitively to the situation when the actor looks around him trying to find some food; suppose now that Find0 fails; in this case,

k) Find(Actor food ABB.objects-perceived) is executed; Find's IS clause is 'primitive'; its **action-for-primitive** is to record the object found; Find1 finds apple1;

l) next, GET0(Actor, apple1) is created and pushed onto **current-path**; this instance's **is** clause consists of Move followed by Take; (in reality, Get has three parameters, the third being the indication of the time that the actor can spend on

retrieving the object -- this is very handy as a precondition if, for example, an adversary can reach the desired object first!)

m) Move0(Actor Apple1) is created and pushed onto **current-path**; Move is a primitive plan, so its **action-for-primitive** asks the user for permission for the actor to move to the point where apple1 is. We grant the permission; Move0 evaluates its **effects**, updating the positions of the actor and all the objects in his inventory and sets its **status** to 'succeeded';

n) **current-path** is appended to **history**; Move0 is popped, and the next plan in the IS clause of Get0 is pushed onto **current-path**: Take0(Actor apple1);

o) Take0 is primitive; its processing is similar to the processing of Move0; it succeeds, one of its effects being that apple1 is added to the actor's inventory, and after manipulations with **current-path** similar to those in l), Ingest0(Actor apple1) is sent to the executor;

p) Ingest is primitive; suppose we allow the actor to ingest the apple; then, after the appropriate (and by now familiar) bookkeeping operations, we find ourselves at the point where Eat0 is proclaimed as succeeded; at this point we evaluate its **effects** and pop it from **current-path** (which at the time contains only PS20, known to have succeeded);

q) **effects** of PS20 are evaluated (the hunger level of the actor is decreased, and a message to this effect is issued), and with this PS20 is popped from **current-path**, which remains empty; this signifies the completion of a cycle of the monitor.

3.2. POPLAR 2.0.

3.2.1. Introduction.

This section describes the changes introduced into the Colgate personality-oriented planner in the new version, POPLAR 2.0. A number of technical improvements were made to support new functionality. POPLAR 2.0 runs on a Symbolics 3600 Lisp Machine. This is a companion text to a previous report on POPLAR: S.Nirenburg , I.Nirenburg and J.Reynolds, POPLAR: A Testbed for Cognitive Modelling, Research Report COSC7, Division of Natural Science, Colgate University, June 1985. This document is structured as follows. First we highlight the additions to the functionality of the system. Next we describe the changes in the knowledge representation introduced in POPLAR 2.0. This is followed by a description of the modified algorithms. Finally, we include a discussion of implementation-related decisions and an example run of POPLAR 2.0. Example plan representations in POPLAR 2.0 can be found in the appendix.

3.2.2. Comparing the functionality of POPLAR 1.3 and POPLAR 2.0.

3.2.2.1. Review of Functionality in POPLAR 1.3.

Before talking about the differences between POPLAR 1.3 and POPLAR 2.0 let us recall the planning algorithm of POPLAR 1.3.

The top level function **Monitor**:

```
begin
 repeat forever
   call the function MAINTAIN-WORLD;
    {which obtains from the user information about changes in the
     world and records it}
   for every object in ABB.objects-perceived† do
      if the object is connected to a certain behavioral need (goal)
      and ABB.agenda* does not contain an instance of a top level
      plan to achieve this goal then create an instance of a top
      level plan to achieve this goal (satisfy this need) and
      add it to the ABB.agenda
   for every physical state in ABB.states-perceived** do
      if ABB.agenda does not contain top level plans for achieving
      corresponding goals (preserve self from hunger, thirst and, or
      fatigue, called the MAINTAIN goals) then create an instance of
      corresponding top level plan and add it to ABB.agenda;
   if ABB.agenda is empty then EXIT (end-repeat);
   for every plan in ABB.agenda do
      produce a rate for this plan by evaluating its rating function;
   choose the plan with the top rating;
   call Executor with the top-rated plan
 end.
```

The function MAINTAIN-WORLD:

```
begin
 repeat
    ask the user whether he/she wants to add objects to the world
    if 'yes' then obtain an object instance name and add it to the
    ABB.objects-perceived
  until the answer is 'no';
  repeat
    ask the user whether he/she wants to remove any object instance
    from the world
```

† ABB objects-perceived is a slot on 'Actor-Blackboard' (see Nirenburg et al., 1985) which holds a list of object instances perceived by Actor at a given time

* ABB agenda (see below) is a slot on 'Actor-Blackboard' which holds a list of top-level plans associated with Actor's goals at the moment

** ABB states-perceived (also see below) is a slot on 'Actor-Blackboard' which holds a list of Actor's physical states (hunger, fatigue, etc.) perceived at a given moment

```
    if 'yes' then obtain an object instance name and delete it from the
    ABB.objects-perceive
  until the answer is 'no';
  repeat
    ask the user whether he/she wants to change any object's
    attributes
    if 'yes' then obtain an object name, names and new values for
    attributes and record them
  until the answer is 'no'
end.
```

The central function of POPLAR is **Executor**. It is described in detail in Nirenburg et al., 1985. Briefly, the algorithm is as follows:

```
begin
 obtain a plan to process
 if the plan's preconditions do not hold
   then report failure and exit
   else expand the plan by considering its is slot:
    if the is clause is 'primitive'
      then perform its Action-for-primitive and
        if was completed successfully then evaluate that plan's effects
      else for every subplan in the is slot
        call Executor with that subplan
 if is slot is processed to its end
    and the last executed subplan was completed successfully
  then report SUCCESS of the current plan,
      evaluate its effects, EXIT;
  else report FAILURE, EXIT;
end.
```

One cycle of the **Monitor** in POPLAR 1.3 covers the choice and the execution of a single top-level plan. This process lasts N time cycles where N is equal to the number of primitive plans involved. (Note that the plan **Move** is also primitive so that it lasts only one time cycle irrespective of the distance between Actor's starting and end positions.)

Changes in the world are made only once during one **Monitor** cycle (at the beginning). This means that no changes obtained during processing are recorded before the beginning of the next **Monitor** cycle. In other words the world remains **monotonic** during one **Monitor** cycle.

In POPLAR 1.3 an attempt was made to approach the solution of this problem in the following way: when a top-level plan is chosen its rating is used to calculate the number of time cycles that this plan will be allowed to execute without interruption. The more 'important' the plan, the longer it is allowed to execute uninterrupted. When an interrupt occurs, the entire **current-path** (a data structure where the stack of executed plans is stored) is sent to **history**. Then the **Monitor** starts a new cycle and if

the same top-level plan is chosen execution starts 'from scratch' even if some of its subplans were completed successfully during the previous **Monitor** cycle.

Both problems described above (the possibility to make the world **nonmonotonic** and ability of saving partial results for the further use) are solved in the new implementation of the POPLAR: POPLAR 2.0.

3.2.3. Improvements in POPLAR 2.0.

A major difference between POPLAR 2.0 and POPLAR 1.3 is the separation of the task of executing an entire top-level plan into two different tasks:

a) execution of non-primitive and 'mental-primitive' plans (this is done by **Executor** in POPLAR 2.0) and

b) execution of 'physical-primitive' plans (this is done by **Effector**) (a detailed description of both algorithms see below in Section 3.1.6.).

The new **Executor** performs a top-level plan execution by expanding its **is** slot - considering its sub-plans (i.e. goes down in the plan hierarchy, or in other words, lowers the level of abstraction). When a 'physical-primitive' plan is encountered control is passed to **Effector** with that plan as a parameter. **Effector** performs a 'physical-primitive' plan execution.

This distinction gives an opportunity in future to perform these two tasks in parallel. The idea behind this decision is that in real life people tend to perform physical and mental actions simultaneously (for example, a person can start scheduling weekend activities while driving to his job on a Thursday).

It is postulated that the execution of a 'physical-primitive' plan lasts one time cycle. Therefore a new **Monitor** cycle now also lasts one time cycle. (Note that the primitive plan **Move** is now represented as one 'step' of the Actor.) So, changes in the world could be perceived by the Actor after every time cycle.

When the top-level plan chosen for execution is already partially executed, **Executor** then starts processing at the point where it left off at the previous step. This point could be found by detecting differences between plan **type** name and plan **instance** name. Subplans that were already processed are presented by plan instance names in the **is** slot.

3.2.4. Plans in POPLAR 2.0.

For the new version changes were introduced into the plan grammar of POPLAR 1.3. (See Figure 5 for the new version of the grammar.) Note that the most important changes were introduced into the plans that involve **finding** objects (e.g., **Fight**, **Eat**, **Drink**). In POPLAR 1.3 these plans contained 'optional' subplans and a decision whether to execute them was made with the help of what we called 'control' functions.

In POPLAR 2.0 instead of both optional paths and control functions a new 'mental-primitive' plan **Get-Selector** is used.

The **get-selector** plan creates new instances of the **Get** plan: one instance for each object instance that was 'found' by the previously performed LOCATE plan. (for

example, Actor is looking for food. Apple1, apple2 and carrot1 are located in the surroundings of the Actor by the plan Locate. Then three instances of the plan **Get** are created: Get1 (object apple1), Get2 (object apple2) and Get3 (object carrot1)). All these instances of the **Get** plan obtain the status 'candidate' and the real-world-flag value of 'no'†. A list of these instances is stored in the **is** slot of the **get-selector** plan for future use. Then one of them is chosen for further execution. The choice is made on the basis of rating. The information on which the rating process is based is contained in the **rating function** slots of the **Get** plans.

When the **Get-Selector** plan is executed for the second or further time, it chooses one of the **Get** plan instances that it had created previously (and stored in its **is** slot). The algorithm for making the choice is as follows: if there is a plan with the status 'suspended', choose *it* for execution, else if there are plans with the status 'candidate' rate them and pick the one with the maximum rate, otherwise report FAILURE.

I ::= P | M | GTR
PS1 ::= FIGHT | HIDE | wait-and-see
PS2 ::= EAT | DRINK | SLEEP
GTR ::= GET
FIGHT ::= FIND move attack
HIDE ::= *locate* move
EAT ::= FIND ingest
DRINK ::= FIND ingest
SLEEP ::= *locate* do-nothing
GET ::= move take
FIND ::= *locate get-selector* GET

Plans shown in lower case are physical-primitive; plans shown in *italics* are mental-primitive.

Vertical bars separate disjoined subplans; in practice, the 'or'-ed plans are chosen on the basis of their ratings through the application of a special 'mental-primitive' metaplan **Plan-Selector**, not shown in the grammar.

Figure 5. A grammar of plans in POPLAR 2.0.

The **Plan-Selector** makes a specific control decision as to which of any possible number of disjoined subplans to choose for fulfilling the current plan. The knowledge for selecting one of disjoined subplans is contained in the **rating-function** slots (methods) of the disjoined subplans.

† The instantiation of plan tokens for possible future use is essentially a way of modeling one-step 'look-ahead' The Actor as if thinks about all possible plans at this point and choose the best of them to perform

3.2.5. The Algorithms.

In this section we present the algorithms of the functions that have been changed in POPLAR 2.0

Executor (P)

a) obtains as parameter a plan, P; P could be either a plan **instance** or a plan **type**[4]. If P is a plan **type then**

   i) create a new instance of this plan

   ii) substitute the plan name P with the name of the new created instance in the **is** slot of the parent plan

   else (i.e. P is a plan **instance**)

   case status

   'failed' exit, reporting failure.

   'succeeded' exit, reporting success[5].

b) check the plan's **preconditions** clause; if preconditions do not hold, (are false) report failure and its reason and exit; otherwise,

c) expand the plan by substituting the contents of its **is** clause for the plan itself:

   if the **is** clause is 'physical-primitive' then exit;

   if the **is** clause is 'mental-primitive' then perform its **Action-for-primitive**. If the current plan is either **Plan-Selector** or **Get-Selector** (metaplans that chose the next plan for execution) and **Action-for-primitive** was performed successfully (a plan is selected) call **Executor** recursively with the selected plan. If **Action-for-primitive** failed then report failure and exit.

   if the IS clause is not 'primitive' then call **Executor** recursively with this plan.

   end **Executor**.

Another new function is **Effector**, whose task it is to monitor actual (simulated) execution of a primitive plan.

The algorithm of **Effector** is as follows:

a) if FAILURE was reported by **Executor** then for each plan in the **current-path** set **status** to 'suspended'; exit;

   else obtain a primitive (physical) plan instance P (from **Executor**). Perform P's **Action-for-primitive**.

b) for each plan in the **current-path** (including P) do: check **satisfaction-condition**; if it holds then perform that plan's **effects** and set **status** to 'succeeded', else set **status** to 'suspended'.

---

[4] note that the agenda holds only plan instances e.g. 'GTR19', whereas **is** clauses of plans can contain both plan **types**, e.g. 'Find', and plan **instances** - in case when those plans were already processed once by **Executor**

[5] This means that plans that were already completed are not executed again

3.2.6. Implementation.

POPLAR 2.0 is implemented in Zetalisp and runs on a Symbolics 3600 Lisp Machine. The new implementation uses the knowledge representation system native to the 3600 Lisp Machine, the Flavors system. Although the overall structure of the knowledge base remains unchanged, a number of internal technical modifications were made.

3.2.6.1. Plan structure.

To maintain the ability to continue execution of a plan from the point it was interrupted, a number of new slots (both instance variables and methods) were added to the basic plan frame (flavor), as specified in the extended EDL of Nirenburg et al., 1985. The syntax of several slots was modified as follows:

WITH
: now contains a list of parameters in the form: ((agent <object-instance-name>) (object <object-instance-name>) (instrument <object-instance-name>) (place <position>)(time-for-execution <time>)

SATISFACTION-CONDITION
: one or more of world states that become true after this plan is executed successfully. This slot is used for understanding the status of the plan, that is, whether the plan is already completed or not.

REAL-WORLD-FLAG
: holds YES when a plan instance is created for immediate execution, and NO if that instance creation is 'look-ahead'. This slot is used by the plan **Get-Selector** (see above).

STATUS
: one of 'on-agenda', 'executed', 'suspended', 'succeeded', 'failed' or 'candidate'.

Appendix 6 contains examples of POPLAR 2.0 plans.

3.2.6.2. The Actor's Blackboard.

Two slots were added to the Actor-Blackboard to help trace Actor's behavior.

CURRENT-GOAL
: holds the Actor's current goal

CURRENT-PLAN
: holds the name of the plan presently executed by Actor.

3.2.6.3. The User Interface.

In POPLAR 2.0 the function MAINTAIN-WORLD which directs the acquisition of information from the user is menu-driven. It contains a menu for choosing an operation (insertion, modification, etc.), a menu for changing an attribute(s) of an object in the world, etc.

The world is represented by the board which contains 200 (10 x 20) mouse sensitive squares. Objects could be moved from place to place or removed from the board using the mouse.

The screen is divided into three separate windows:

a) the Actor's world (see above);

b) a LISP Listener window where all messages generated by the program appear

c) the TRACE window which monitors the current values of the most important parameters and data structures including

   i) the CLOCK of the system

   ii) Actor's blackboard which contains AGENDA of goals, a current plan, a current goal as well as

   iii) Actor's physical state parameters.

   All the values are updated immediately after changes were introduced.

3.2.7. An Example.

Suppose we want to test POPLAR's performance in the following world situation. The room will contain, in addition to the Actor itself, a rock, a snake, a sword, a dog and a stick. The positions of all the objects will be as shown in Figure 6. Recollect that the world in which POPLAR operates at the moment is that of an actor in a room with sources of danger, food and treasure. The actor is 'programmed' to plan survival and maintenance of self plus getting as much of the treasure as possible in its possession.

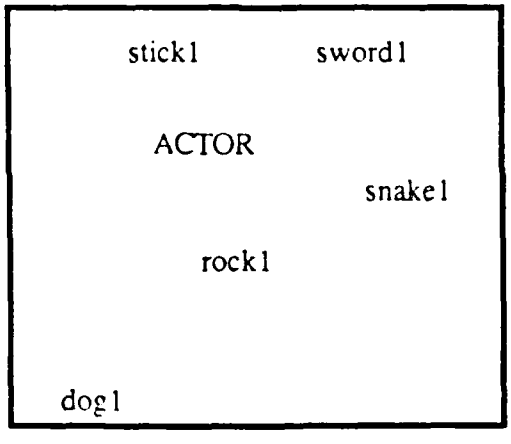


Figure 6. An Instance of the POPLAR World

POPLAR 2.0 acts as follows:

a) obtains a new input from the user:

   i) first it produces a menu containing the list of possible operations of the following types:
   creating new object instances,
   adding existing object instances to the world or
   changing attributes of the object instances in the world.
   The user clicks on 'creating new object instances', then

   ii) the next menu containing the list of all object types that are predefined in the system is popped.

   The user chooses, say, 'snake'. Then

   iii) the menu containing the list of the parameters for 'snake', with default values, is popped, so that the user can make changes there.

   The procedure iterates until specifically told to exit, so that the user inserts all desired objects into the world.

   POPLAR 2.0

b) processes the new input: since snakes and dogs spawn the need for protection, two instances of the goal Protect (P1 and P2) are created and added to ABB.agenda.

c) checks Actor's physical state (hunger, etc.). Suppose that none of the hunger, thirst and fatigue is above the detecting threshold, therefore, ABB.states-perceived is empty.

d) the function Agenda-Scheduler is called. The goal P1 ((agent Actor)(object snake1)) gets the highest rate because the Actor (as it is adjusted for this particular run of POPLAR) is more concerned about snakes than about dogs. Note also that the snake is closer to the Actor than the dog.

e) the **Executor** is called with P1 which is now posted in both ABB.current goal and ABB.current-plan. P doesn't have any preconditions, so its **is** clause is expanded: first a new instance of **Plan-Selector** is created. Since **Plan-Selector** is a 'mental-primitive' plan, its **Action-for-Primitive** is performed:

f) new instances of the plans **Fight**, **Hide** and **Wait-and-See** are created and rated (using their rating functions). Fight1 is selected. ABB.current-path now contains (P1 Fight1).

g) **Executor** is called with Fight1. After checking the preconditions of Fight1 (there are none) the IS clause is expanded to (Find Move Attack). Find1 with ((agent Actor)(object weapon)) is created and pushed onto ABB.current-path. Then Find1 is in its turn expanded to (Locate Get-selector Get).

h) Locate1 with ((agent Actor)(object weapon)) is created and since it is 'mental-primitive', its **Action-for-primitive** is performed. The result is the list of objects 'found' (stick1 sword1). (Actor 'knows' that both sticks and swords could be used as weapons against snakes). The plan Locate1 is completed successfully.

i) Get-selector1 is created and its **Action-for-primitive** is performed. As the result of this two instances of **Get** (Get1 ((agent Actor)(object stick1)) and Get2 ((agent Actor)(object sword1)) are created and rated. The **Get** with the highest rate (in this case Get2 because sword is more effective weapon then stick) is selected for execution.

j) **Executor** is called with Get2 which is expanded to (Move Take).

k) **Executor** is called with **Move**; Move1 is created. Since its **is** slot is 'physical-primitive' control is passed to **Effector**.

l) **Effector** performs Move1's **Action-for-primitive** which is 'make one step towards the position of sword1'. This step is made, i.e. the position of Actor is changed. Then the SATISFACTION-CONDITION1 of all plans in the ABB.current-path are checked (the current path at this point contains (Move1 Get2 Fight1 P1). None is satisfied, so the values of their 'status' slots are set to 'suspended'.

m) a new cycle of the **Monitor** begins. The user has again an opportunity of changing the world. Suppose, the user does not want to change anything. So, the agenda of goals remains the same and the top rated goal is the same P1.

n) the path of 'suspended' plans is found (Move1 Get2 Find1 Fight1 P1). Since Move1 is not yet completed, control is passed to the **Effector** with Move1.

o) the **Effector** performs the next 'step' of the Actor's movement towards sword1. Then SATISFACTION-CONDITIONs of all plans in the ABB.current-path are checked, none is satisfied, so the values of their 'status' slots are set to 'suspended'.

p) a new cycle of the **Monitor** begins.

4. Background and Related Work.

In designing and implementing POPLAR 2.0 a number of conceptual and technical decisions and choices had to be made. The following is an incomplete, though representative list.

1) how does one approach, and justify, construction of a multi-faceted system when little is known about the peculiarities of its components? Where is the starting point?

2) how might the problem of personality influences upon cognition be addressed?

3) within cognitive component, how are goals and plans related? How are they each related to such concepts as needs, drives, performance, etc.?

4) What is the structure of the planning module in cognitive systems? How is the scheduling of the cognitive system's activities performed?

5) What is the relationship between the use of internalized (canned) and newly created plans?

6) What is the relation between plan production and plan understanding?

The realization of the above and some additional problems was instrumental in the design stage. While not all of the decisions have been already made at this stage, our desire was to avoid design choices that would preclude or hamper a future improvement or extension.

None of the theoretical or design decisions were made without the influence of other, previous, related work. In this section we briefly review the bases for the various decisions as well as mention other work on the problems we faced.

Fundamental to the development of POPLAR was the approach to the task, faced by most cognitive modelers, of building a structure consisting of a number of distinct constituents, the details of many of which were (and at present remain) unknown. How does one construct a global model when many of its components are uncertain, and each one is itself a mystery? Here we adopted the attitudes advocated by Haugeland (1981), who suggests that it is appropriate to study an entire information processing system (IPS), consisting of several modules each of which (plus the IPS itself) is a black box, without first completing the study of the components; thus, we studied the cognitive actor even though we had not (and, obviously, could not) first provided an account for perception and performance.

Norman (1981) was very instrumental in specifying the tasks to be tackled in cognitive modeling. We also owe much to Anderson's (e.g. 1983) work on the architecture of cognitive entities. Sloman & Croucher (1981) discuss the introduction of motives, moods, attitudes and emotions in natural and artificial intelligent systems. Although no formalism is suggested for encoding this type of information, the general thrust of the approach is valuable for those who consider the introduction of certain personality characteristics into a class of AI systems. Wallace (1981) addresses similar problems in the context of learning.

Uhr & Kochen (1969) is an early work that addressed similar issues. Many of the important points for POPLAR have been anticipated in that work. Unfortunately, Uhr & Kochen's approach cannot be even called knowledge-based. It was an attempt to perform an important piece of research with inadequate means.

Wood (1983) discusses planning in a dynamically changing world with multiple actors. Her system, AUTODRIVE, uses the world of the automobile driver as the domain. Although the design of the system depends too strongly on the implementation world, the idea of interaction between the actor and the world (in fact, the mere separation of the objective world and that of the actor -- through a program called SIMULATOR) is very fruitful.

Schank & Abelson (1977) and Schank & Lehnert (1979) informally discuss and catalog human (including interpersonal) goals. Carbonell (e.g. 1979) discusses the use of the concept of personal goals in the context of understanding stories. Wilensky (1983) also discusses everyday goals and metagoals, as well as various cooperative and competitive relations among them.

The relation between goals and plans is an interesting question that had to be addressed in our work. Our solution was to use this term only for top-level goals recognized by the goal-generators but made manifest in the system through the instantiation

of a top-level plan. We did not use the concept of goals at lower levels in planning (i.e. we did not use the term 'subgoaling', cf. Lesk, 1984).

It is argued (cf. e.g. Barber (1983) or Berlin (1984)) that subgoaling is preferable to the use of 'canned' plans because if the latter are used then there is no possibility of ever achieving a goal in a non-standard way. But in the subgoaling approach, within the current state of the art, no unexpected results can be obtained either. To introduce these, one has to build a learning system, one capable of creating and not only recreating. But at present the planning of the subgoaling type remains no less 'canned' than the the 'forward' planning.

It seems that these two approaches to planning relate essentially in the same manner in which backward chaining relates to forward chaining in inference making. Our opinion is that the choice between the two is not strategically important and should reflect the peculiarities of the domain and other 'weak' considerations, so typical for AI.

Another important issue related to goals and plans is whether to build systems that in scheduling an action take into consideration the knowledge of how many different plans and or goals will be furthered by it. The main empirical body of Wilensky's book (1983) is devoted to such issues. Cf. also Hammond (1983) for a philosophically related approach. Hayes-Roth & Hayes-Roth (1979) also want their planner to have this capability. Our position on this topic (cf. also Carver et al., 1984) is that in the type of planners we are building the goal cooperation or conflict does not play a role. We argue that to treat this topic as central in modeling planning in intelligent actors is similar to consider such non-everyday tasks as playing chess and solving differential equations central topics for AI. The latter methodological fallacy has been amply criticized.

General works on planning that immediately influenced this project include Stefik's work (e.g. 1981) on metaplanning and planner architecture. Hayes-Roth & Hayes-Roth (1979) describe a very rich planning domain and offer a good discussion of what the editors of **The Handbook of AI** (Cohen & Feigenbaum, 1982, p. 519; cf. also pp. 22 - 27) call opportunistic planning. It does not seem, however, that a non-trivial, involved implementation of the itinerary planner they suggest is possible.

Hayes-Roth (1984) is a definitive proposal concerning the architecture for planners. It addresses the control problem in AI systems as a whole. It also contains a comparison with other current proposals concerning control. In its architectural part this proposal (in fact, not only this proposal!) draws heavily on the earlier work in the HEARSAY-II speech understanding system that introduced and popularized the blackboard architecture (cf. Erman et al., 1980).

The crucial idea of metalevel reasoning is discussed, with different emphases, in Stefik (1981), Hayes-Roth (1984), Wilensky (1983) and Genesereth (1983).

The basic architecture of POPLAR has a number of common points with that of Wilensky's planner (cf. Wilensky, 1983, pp.22-23). The two models, however, display major differences, notably in the attention paid in POPLAR to the problem of scheduling or in importance attributed to the idea of the independent representation of the

objective world. Insufficient attention to scheduling and to describing the planning process at the system level were prominent among the criticisms in some reviews of Wilensky's book (cf. Russell, 1984; Berlin, 1984).

Many interesting ideas about scheduling can be found in Sathi et al. (1984) and Fox (1983).

Work on understanding plans in the POISE (Croft et al., 1983) and Argot (e.g. Litman & Allen, 1984) projects has helped in formulating some parts of our approach.

5. Status and Future Development.

5.1. General.

POPLAR is a working system that generates and executes a relatively small number of plans in a rich, though simulated, environment. Its scheduling capabilities actually seem to transcend the immediate necessities of the domain. The system is designed in such a way that both domain planning and metaplanning are performed by one executor (that is, POPLAR can reason about its own actions). A number of features have been included that make POPLAR a model of a human planner in a real world.

At the same time, the possibilities of development and improvement that this basic system offers are probably even more exciting than experimentation with the current version of POPLAR. There are many points at which the system can be improved. Some of them are discussed below.

First (and simplest) of all, the POPLAR actor's knowledge about the objects, goals and processes both in the objective world and its own 'mind' can and will be augmented. In parallel, the control knowledge (rating and control functions) will be constantly adjusted and tuned, both through the introduction of additional character trait, mental state and situation parameters and through devising more appropriate ways of amalgamating them in the decision functions. Extensive experimentation with POPLAR will help to verify such decisions.

In parallel and in conjunction with the POPLAR project, these authors have been involved in designing a general model of human cognitive activity. Initial results of that research are reported elsewhere (Nirenburg & Reynolds, 1983; Reynolds & Nirenburg, in preparation). An aspect of that project extremely helpful to POPLAR is research aimed at deriving a set of 'primitive' character traits, motivations and mental states, such that weighted combinations of them will correspond to the 'higher-level' parameters (e.g. 'aggressiveness') that we would like to use in POPLAR's decision functions.

One can see that the above are actually two separate problems: 1) to extract primitives; 2) to express complex entities in terms of the primitives. It was decided to adapt the primitives suggested by Cattell (cf. e.g. Cattell & Child, 1975). Extensive psychological experimentation with humans is pursued in order to find answers to the second problem. The benefits of having a system that boasts psychologically valid (and not 'folk psychology'-based) control parameters are enormous and self-evident. And, therefore, this is one of the most immediate improvements we plan to make.

We also plan to add plan understanding to plan production. The world is inhabited by more than one cognitive actor (consider, for instance, the trolls in the current POPLAR). In order to behave correctly an actor must be able to discern plans of others. We believe that POPLAR's machinery will be able to handle plan recognition without the necessity to introduce major changes. An actor will maintain as many blackboards as there are cognitive actors around. It will assume that all other actors operate in the same manner. It will have beliefs about their character traits, etc. and will 'project' plans for them much in the same manner as it plans.

A logical extension to adding plan recognition is to introduce verbal behavior into POPLAR. There exist a number of interesting approaches to discourse analysis and plan understanding in dialogues (e.g. Allen, 1983b; Litman & Allen, 1984; Carberry, 1983; Reichman, 1984; etc.). A study in modifying POPLAR to involve verbal behavior and discourse analysis can be found in Nirenburg & Pustejovsky (1985).

The inclusion of multiple actors into the objective world can lead to the development of an experimental testbed for modeling conflict resolution, cooperation and many more important 'real-life' situations. The possibilities here are definitely substantial and quite unexplored.

The mechanism for modeling attention will undergo serious modifications, as will the treatment of time and the interaction between the actor(s) and the objective world.

And, finally, a most important avenue of improvement is the introduction of learning capabilities to the system. There are many modules in POPLAR where planning can be introduced; and there are many different types of learning to be studied. Some examples of this may be modifying the scheduling behavior depending on results of previous processing or after seeing somebody achieve a goal in a way not previously used, modifying beliefs about objects; being able to 'create' new plans, by analogy or otherwise; and many many more. This topic is one of the more complex ones, but any progress in this direction may have a very beneficial effect on the field of planning in AI.

5.2. Specific Plans: POPLAR 3.0.

The development plans for the coming year include the development and implementation of

1) strategies for combining plan **recognition** and plan **production** in one system,
2) a **mixed strategy** for planning: the use of canned plans for standard situations and 'first principles' knowledge when non-standard situations arise;
3) the development of a model of (a subset of) the world of I & W.

Bibliography.

Allen, J., 1983a. Maintaining knowledge about temporal intervals. *Communications of ACM*, vol. 26, 832 - 843.

Allen, J., 1983b. Recognizing intentions from natural language utterances. In: M. Brady & R.C. Berwick, eds., **Computational Models of Discourse.** Cambridge, MA: MIT Press.

Anderson, J.R., 1983. **The Architecture of Cognition.** Cambridge MA: Harvard University Press.

Barber, G., 1983. Supporting organizational problem solving with a work station. *ASM Transactions on Office Automation Systems*. Vol.1, No.1, January. 45 - 67.

Bates, P., J. Wileden and V. Lesser, 1981. A language to support debugging in distributed systems. University of Massachusetts COINS Technical Report 81-17.

Berlin, Daniel, 1984. Review of Wilensky (1983). *Artificial Intelligence*, vol. 23, 242-244.

Carberry, S., 1983. Tracking user goals in an information-seeking environment. Proceedings of AAAI-83. Washington, DC.

Carbonell, J., 1979. Computer models of human personality traits. Proceedings of IJCAI-79. 121 - 123.

Carver, Norman F., Victor R. Lesser and Daniel L. McCue, 1984. Focusing in plan recognition. Proceedings of AAAI-84. Austin, TX.

Cattell, R.B. and D. Child, 1975. **Motivation and Dynamic Structure.** NY: Wiley.

Cohen, P.R. and E.A. Feigenbaum (eds.), 1982. **The Handbook of Artificial Intelligence.** Volume III. Los Altos CA: Kaufmann.

Croft, W.B. and L. Lefkowitz, 1984. Task support in an office system. *ACM Transactions on Office Automation Systems*. Vol.2, No.3 197 - 212.

Croft, W.B., L. Lefkowitz, V. Lesser and K. Huff, 1983. POISE: An intelligent interface for profession-based systems. Conference on Artificial Intelligence. Oakland, Michigan, 1983.

Deering, M., J. Faletti and R. Wilensky, 1981. PEARL - a package for efficient access to representations in Lisp. Proceedings of 7th IJCAI. Vancouver, BC. 930 - 932.

Erman, L.D., F. Hayes-Roth, V.R. Lesser and D.R. Reddy, 1980. The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty. *Computing Surveys*, vol. 12, 213-253.

Fox, M., 1983. Constraint-directed search: a case study of job-shop scheduling. CMU Robotics Institute Technical Report 83-22.

Genesereth, M.R., 1983. An overview of meta-level architecture. Proceedings of AAAI-83. Washington, DC.

Hammond, K.J., 1983. Planning and goal interaction: the use of the past solutions in present situations. Proceedings of AAAI-83, Washington, DC.

Haugeland, J., 1981. The nature and plausibility of cognitivism. In: J. Haugeland (ed.). **The Mind Design.** Cambridge MA: MIT Press.

Hayes-Roth, B., 1984. A blackboard model of control. Stanford University Heuristic Programming Project Report HPP 83-38 (revised August 1984).

Hayes-Roth, B and F. Hayes-Roth, 1979. A cognitive model of planning. Cognitive Science, vol. 3, No.4.

Lesk, M., 1984. Universal Subgoaling. CMU PhD Thesis.

Litman, D. and J.Allen, 1984. A plan recognition model for clarification subdialogues. Proceedings of COLING-84. Stanford, 302 - 310.

M Cue, Daniel and Victor Lesser, 1983. Focusing and Constraint Management in Intelligent Interface Design. University of Massachusetts COINS Technical Report 83-36.

Nirenburg, S. and J. Pustejovsky, 1985. Plan Recognition and Production for Verbal (Discourse) and Non-Verbal Behavior. COINS Technical Report, University of Massachusetts.

Nirenburg, S. and J.H. Reynolds, 1983. An architecture for a computer model of human cognitive systems. Colgate University COSC Technical Report 4-83.

Norman, D., 1981. Twelve issues for cognitive science. *Cognitive Science*, vol. 4, No.1.

Reichman-Adar, R. Extended person-machine interface. *Artificial Intelligence*, vol.23, 157 - 218.

Reynolds, J.H. and S.Nirenburg, in preparation. The relationship between motivational traits, goal generation and plan selection.

Russel, Daniel.M., 1984. Review of Wilensky (1983). *Artificial Intelligence*. vol. 23, 239-242.

Sathi, A., M. Fox, M. Greenberg and T. Morton, 1984. Callisto: an intelligent project management system. CMU CS working paper.

Schank, R.C. and R. Abelson, **1977. Scripts, Plans, Goals and Understanding**. Hillsdale, NJ: Erlbaum.

Schank, R.C. and Lehnert, W., 1979. The conceptual content of conversations. Proceedings of 6th IJCAI. 769 - 771.

Sloman, A. and M. Croucher, 1981. Why robots will have emotions. Proceedings of 7th IJCAI. Vancouver, BC, 197-202.

Stefik, M, 1981. Planning with constraints (MOLGEN: Part 1 and Part 2). *Artificial Intelligence* 16, 111-170.

Uhr, L. and M. Kochen, MIKROKOSMS and robots. Proceedings of IJCAI-69, Washington, D.C., 541 - 556.

Wallace, J.G., 1981. Motives and emotions in a general learning system. Proceedings of 7th IJCAI. Vancouver, BC, 84-86.

Wilensky, Robert, 1983. **Planning and Understanding**. Reading, MA: Addison-Wesley.

Wood, S., 1983. Dynamic world simulation for planning with multiple agents. Proceedings of IJCAI-83, Karlsruhe, Germany.

Appendix 1. Representation of OBJECTS in POPLAR 1.3

We present objects in two ways: first, in the way the object is stored in LTM, and second, from within a POPLAR run (as an annotated script). The difference is due to inheritance of parents' properties by children in the hierarchy.

A.

```
(dbcr exp creature person ;this is a PEARL header for a frame
    (id person)
    (type creature)    ;CREATURE is the parent of PERSON
    (h-process-roles  lisp ((Take Who)
                            (Put Who)
                           (Find Who)))
                      ;the above are the roles in which an instance
                      ;of this type can appear in specified
                      ;processes by virtue of its having properties
                      ;of a "human": humans can act as agents in
                      ;TAKE, PUT and FIND
    (mental-state struct) ;humans have mental states -- cf. the
                          ;default values in the script listing below
    (character-traits struct char-traits) ;ditto
    (weapon-against ((sword 100 3)(knife 50 1)(rock 10 20)))
          :POPLAR knows (believes) that weapons against people
          ;include swords, knives and rocks; the numbers (a b)
          ;indicate the efficiency of the weapon and the maximum
          ;range
    (power 50) ;maximum
    (speed 50) ;maximum
    (fearsomeness 25) ;what is the level of fear that such objects
                      ;typically elicit in POPLAR (default: 25)
    (mass 55)
    (inventory lisp) ;the objects this person is perceived by POPLAR
                     ;to be carrying
)
```

B.

```
POPLAR > person
  (person (id person)
        (type creature)
;-------------------------------------------------
        (c-process-roles ((Find What)))
;this property is inherited by virtue of PERSON's being a
;descendant of OBJECTS: any object can occupy the "what" slot in Find,
;because finding mental objects is recollecting their representations in
;memory
;-------------------------------------------------
        (shape nil)
        (color nil)
        (mass 55)
        (position nil)
        (p-process-roles ((Take What) (Put What)))
        (goal-parameters ((PS1 adv)))
;the above properties are inherited by virtue of person being a descendant
;of PHYSICAL-OBJECTS; the goal-parameters slot specifies an instance of
;what goal is created when an object of this type is perceived. In this
;case the intuition behind the entry is that the appearance of a person
;spawns the creation of a goal instance of Preserve-Self-1, that is,
;persons are perceived by POPLAR as potential enemies
;-------------------------------------------------
        (edibility nil)
;this property is inherited by virtue of PERSON's being a descendant of
;+alive; nil is the default value with the semantics of "unknown"
;-------------------------------------------------
        (c-process-roles
         ((Eat Who)
          (Ingest Who)
          (Drink Who)
          (Move Who)
          (Attack (Who Whom))))
;the above properties are inherited by virtue of PERSON's being a
;descendant of CREATURE; creatures are considered by POPLAR to be able
;to be agents of eating, drinking and moving, and agents and objects of
;attacking
;-------------------------------------------------
        (weapon-against ((sword 100 3) (knife 50 1) (rock 10 20)))
        (power 50)
        (fearsomeness 25)
        (speed 50)
        (orientation nil) ;this shows whether this particular person
                          ;LOOKS at POPLAR at the moment of processing
```

```
;-----------------------------------------------------
;the following are physical states (conceptually, they are a part of
;the regulatory system)
        (hunger 0)
        (thirst 0)
        (fatigue 0)
        (injury 0)
        (h-process-roles ((Take Who) (Put Who) (Find Who)))
        (mental-state (nilstruct))
;character traits are a component of the regulatory system
        (character-traits
         (char-traits (greed 20)
                      (pedantism 10)
                      (hunger-tolerance 5)
                      (thirst-tolerance 20)
                      (fatigue-tolerance 20)
                      (courage 25)
                      (aggression 40)
                      (impulsiveness 30)
                      (articulateness 40)
                      (extravertedness 50)
                      (loquaciousness 40)
                      (curiosity 55)))
        (inventory nil))
```

Appendix 2. Examples of PLAN representation in POPLAR 1.3.

```
(dbcr exp PLANS PS1
  (ID PS1)
  (Type PLANS)
  (Top-level-flag yes)
  (IS ((Plan-Selector  Fight Wait-and-See))) ;Flee Hide
  (With   (Actor Adversary))
  (COND  ((Plan-Selector '(Fight Wait-and-See)   ;Hide Flee
                    current-plan)
        (Fight Actor Adversary)
        (Flee Actor Adversary)
        (Hide Actor Adversary)
        (Wait-and-See Actor Adversary)))
  (Preconditions (and (member 'Adversary (getpath ABB '(OBJECTS-PERCEIVED)))
          ;Adversary is among objects perceived by the Actor
                (or (= 'Actor 'self)
                   (and (structurep Actor)
                        (not (structurenamep 'Actor))
                        (= (getpath (eval Actor) '(type) 'person))))))
         ;Actor is either "self" or any instance of person
  (Rating-function (rating-func-PS1)))
```

```
(dbcr exp PLANS Plan-Selector
  (ID Plan-Selector)
  (Type PLANS)
  (Top-level-flag no)
  (IS (primitive))
  (Action-for-primitive (Schedule-of-plan 'list-of-plans 'calling-plan))
  (With (list-of-plans calling-plan))
  (Time 1) )

(dbcr exp PLANS Fight
  (ID Fight)
  (Type PLANS)
  (Top-level-flag no)
  (IS (Find (Control1 (Find ' (Control2 (Get))) (Control3 (Move ' Attack))))
  (COND ((Find  Actor (getpath  Adversary '(weapon-against))
           (getpath Actor '(inventory)))
         (Find  Actor (getpath  Adversary '(weapon-against))
           (getpath ABB '(OBJECTS-PERCEIVED)))
         (Get Actor  (car result-find)
           (div (distance Adversary (car result-find))
             (getpath Adversary '(speed))))
         (Move Actor (prog (weapon-range)
           ;position to Move to
           (cond ((<= (distance Actor Adversary)
                      (setq weapon-range
                        (caddr (assoc (getpath (eval (car result-find))
                                              '(type))
                                      (getpath (eval Adversary)
                                              '(weapon-against))))))
                 ;if distance between Actor and Adversary is less
                 ;(or equal) than the range of the Actor's weapon
                 ;then Actor doesn't need to move towards Adversary
                  (return (getpath (eval Actor) '(position))))
                 (t (return (calculate-position Actor
                                                Adversary weapon-range)))
           )))

         (Attack Actor Adversary (car result-find))))

  (Control ((Control1 (Fight-Control1 Actor Adversary))
            (Control2 (Fight-Control2 Adversary))
            (Control3 (Fight-Control3))))
  (With  (Actor Adversary))
  (Rating-function (rating-func-fight)) )
```

```
(defun Fight-Control1 (Actor Adversary)
    (cond ((not (= (car ABB CURRENT-PATH Status)
                   'succeeded))
           t)
   ;EITHER the last executed plan (which is Find) failed
          ((= (cadar Adversary.weapon-against)
              (cadr (assoc (car result-find).type
                           Adversary.weapon-against)))
           nil)
   ;OR actor's current weapon is NOT the most efficient weapon
   ; against this adversary
          ((lessp
               (div (times (distance Actor Adversary)
                           (diff (cadar Adversary.weapon-against)
                                 (cadr (assoc (car result-find).type
                                              Adversary.weapon-against))))
                    Actor.character-traits.impulsiveness)
               fight-control1-threshold))
   ;OR even if the actor does not have the best weapon, he may decide not to
   ;look for a better one -- if the distance between him and the adversary
   ;is too small, if the actor is very impulsive or if the weapon is not
   ;much worse than the best one


(defun Fight-Control2 (Adversary)
 (cond ((null (cadr result-find)) t)
   ;no weapon was found in actor's possession
     ((greaterp (cadr (assoc (car result-find).type
                             Adversary.weapon-against))
                (cadr (assoc (cadr result-find).type
                             Adversary.weapon-against))))
   ;the weapon that was found "around" is BETTER than
   ;the weapon in actor's possession
```

Appendix 3. Examples of POPLAR 1.3 rating functions.

A. The rating function for the Preserve-Self-1 goal (and top-level plan)

```
(defun rating-func-PS1 (actor adversary)

  (fix (div
         (times
              (calculate-fear actor adversary)
              actor.aggression)
         actor.courage)))

(defun calculate-fear (actor adversary)

  (fix (div
        (times adversary.orientation
               (add adversary.mass adversary.speed)
               adversary.power
               adversary.aggr
               adversary.fearsomeness)
        (times (fix (add1 (log (distance actor adversary))))
               actor.courage
               actor.power
               (add actor.mass actor.speed)))))
```

B. The rating function for the Fight intermediate plan.

```
(defun rating-func-fight (actor adversary)

 (fix (div
         (times adversary.weapon-against.efficiency
                actor.courage
                actor.power
                (add1 adversary.injury)
                (expt actor.aggression 2))
         (times
                (calculate-fear actor adversary)
                adversary.power
                (add1 actor.injury)
                adversary.fearsomeness
                (add1 actor.fatigue)))))
```

Appendix 4. HISTORY in POPLAR 1.3.

;this is the way HISTORY looks at the end of the example run of 5.4.

```
POPLAR> HISTORY
  ((Ingest0 Eat0 PS20)
   (Take0 Get0 Eat0 PS20)
   (Move0 Get0 Eat0 PS20)
   (Get0 Eat0 PS20)
   (Find1 Eat0 PS20)
   (Find0 Eat0 PS20)
   (Eat0 PS20)
   (Plan-Selector0 PS20))
```

Appendix 5. BLACKBOARDS in POPLAR 1.3.

Typical contents of the worldand the actor blackboards.

```
POPLAR> WBB
      (World-Blackboard (ID WBB)
                  (NEW-INPUTS troll1 apple2 crocodile2)
                  (TIME (Base-Time (ID Time)(act-time 17))))

POPLAR> ABB
      (Actor-Blackboard (ID ABB)
                  (OBJECTS-PERCEIVED (troll2 sword1 gold-nugget2))
                  (STATES-PERCEIVED (hunger fatigue))
                  (AGENDA PS14 PS22 GTR4 Agenda-Scheduler)
                  (CURRENT-PATH (find7 fight3 PS14)))
```

Appendix 6. Examples of PLAN representation in POPLAR 2.0.

```
(defflavor Plans (id
                  is
                  top-level-flag
                  with
                  status
                  satisfaction-cond
                  param-binding
                  real-world-flag)
   ()
 :settable-instance-variables
 :gettable-instance-variables
 :initable-instance-variables
)

(defflavor P ((id P)
              (is '((Plan-Selector Fight Flee Hide Wait-and-See)))
              (top-level-flag 'yes)
              (with '((agent actor)(object adversary)(instrument weapon)))
              (status 'init))
   (Plans)
 :settable-instance-variables
 :gettable-instance-variables
 :initable-instance-variables
)

(defmethod (P :init) (options)
  (setq param-binding '((Plan-Selector (Fight Wait-and-See Hide) .id)
                        (Fight .(assoc 'agent with)
                               .(assoc 'object with)
                        (weapon nil))
                        (Wait-and-See .(assoc 'agent with))
                        (Hide  .(assoc 'agent with)
                               .(assoc 'object with)))
        satisfaction-cond '(not (member (quote .(cadr (assoc 'object with)))
                                        (send ABB :objects-perceived))))
        ;goal P is achieved when Adversary is not among
        :the objects perceived by Actor
)

(defmethod (P :rating-function) ()
 (let* ((actor (eval (assoc 'agent with)))
        (adversary (eval (assoc 'object with)))
        (dist (distance actor adversary))
```

```
      (adv-mass (send  adversary :mass))
      (actor-mass (send  actor :mass))
      (orientation (cond ((equal (send  adversary :orientation) 'yes)
                          2)
                         (t 1)))
      (adv-power (send  adversary :power))
      (actor-power (send actor :power))
      (actor-courage (send  actor :courage))
      (adv-speed (send  adversary :speed))
      (actor-speed (send  actor :speed))
      (adv-aggr (send  adversary :aggression))
      (actor-aggr (send  actor :aggression)))

  (fix (div (times orientation
                   (add adv-mass adv-speed)
                   adv-power
                   adv-aggr
                   actor-aggr
                   (send  adversary :fearsomeness))
            (times (fix (add1 (log dist)))
                   actor-courage
                   actor-courage
                   actor-power
                   (add actor-mass actor-speed)))
  )
)


(defflavor Find ((id Find)
                 (is  '(Locate (Plan-Selector Get)))
                 (top-level-flag 'no)
                 (with '((agent actor)(object obj)))
                 (status 'init))
   (Plans)
 :settable-instance-variables
 :gettable-instance-variables
 :initable-instance-variables
)


(defmethod (Find :init) (options)
  (setq param-binding '((Locate .(assoc 'agent with)
                                .(assoc 'object with))
                        (Plan-Selector (Get) .id)
                        (Get  .(assoc 'agent with)
```

```
                    (object obj)))
      satisfaction-cond '(member (cadr (assoc 'object with)))
                          (send (eval (cadr (assoc 'agent with)))
                              :inventory))
      ;goal Find is achieved when Actor has Object
      ;in his possession
)

(defmethod (Find :preconditions) ()
  (cadr (assoc 'object with))
  ;the object is specified (Actor 'knows' what is to be found)
)


(defflavor Take (tid Take)
            (is '(physical-primitive))
            (top-level-flag 'no)
            (with '((agent actor)(object obj)))
            (status 'init))
   (Plans)
 :settable-instance-variables
 :gettable-instance-variables
 :initable-instance-variables
)


(defmethod (Take :init) (options)
  (setq satisfaction-cond '(member (cadr (assoc 'object with)))
                          (send (eval (cadr (assoc 'agent with)))
                              :inventory))
      ;goal Take is achieved when Actor has Object
      ;in his possession
)

(defmethod (Take :preconditions) ()
  (and (member (cadr (assoc 'object with))
           (send ABB :objects-perceived))
     ;Object is among the objects perceived by Actor
      (equal (send (eval (cadr (assoc 'agent with))) 'position)
           (send (eval (cadr (assoc 'object with))) :position)))
     ;Object and Actor are on the same position
)

(defmethod (Take :action-for-primitive) ()
```

```
 ;the part of the code that maintains the graphic
 ;part of the program is omitted

(let ((actor  (cadr (assoc 'agent with)))
      (object (cadr (assoc 'object with))))
 (send  (eval actor) :set-inventory
        (cons object
              (send (eval actor) :inventory)))
   ;add Object to Actor's inventory
 (format t "~&~a ~a ~a ~a ~&"
         object
         "is now in"
         actor
         "'s possession.")
 (send ABB :set-objects-perceived
       (delete object (send ABB :objects-perceived)))
 )
)
```

PROVIDING INTELLIGENT ASSISTANCE IN DISTRIBUTED OFFICE ENVIRONMENTS[1]

Sergei Nirenburg
Colgate University

Victor Lesser
University of Massachusetts

Abstract. We argue that a task-centered, an agent-centered and a cognition-oriented perspective are all needed for providing intelligent assistance in distributed office environments. We present the architecture for a system called OFFICE that combines these three perspectives. We illustrate this architecture through an example.

1. Introduction.

In this paper we describe OFFICE, a system that provides intelligent assistance in the office environment. A schematic diagram of the type of system we are proposing is shown in Figure 1.

In this diagram the office worker operating together with his/her workstation constitute one node in the office problem solving network. The initiative in such a problem-solving environment is mixed: it can be originated by the office worker performing a low-level task or specifying a high-level goal to be accomplished or the office system OFFICE requesting the worker to perform a task. Thus, we see OFFICE as an intelligent assistant to the office worker.

We argue that a task-centered, an agent-centered and a cognition-oriented perspective are all needed for providing intelligent assistance in distributed office environments. We need knowledge from each of these perspectives in order to support not only effective local interaction between OFFICE and the office worker, but also to coordinate cooperative problem solving among the nodes in the system. Coordinating problem solving is an especially difficult task, given the semi-autonomous nature of processing at each node; the bandwidth of the communication channel (which makes it not feasible for nodes to have a complete global view of problem solving in the network); the diversity of the types of knowledge necessary for coordinating and scheduling office activities; and the necessity to provide guidance to the office worker about how to prioritize his own tasks so that they are coherent with the goals of the whole system.

[1] This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY 13441-5700, and the Air Force Office of Scientific Research, Bolling Air Force Base, D.C. 20332 under contract number F30602-85-C-0008.

We see the coordination problem as breaking down into a number of subproblems, which include managing resources; equalizing workload distribution; managing goal conflicts; maintaining a proper level of redundancy in task execution and especially in information flow; analyzing dependencies in the sets of goals, plans and events, etc. Automation of any of the above tasks clearly involves manipulation of many types of knowledge, both domain and control.

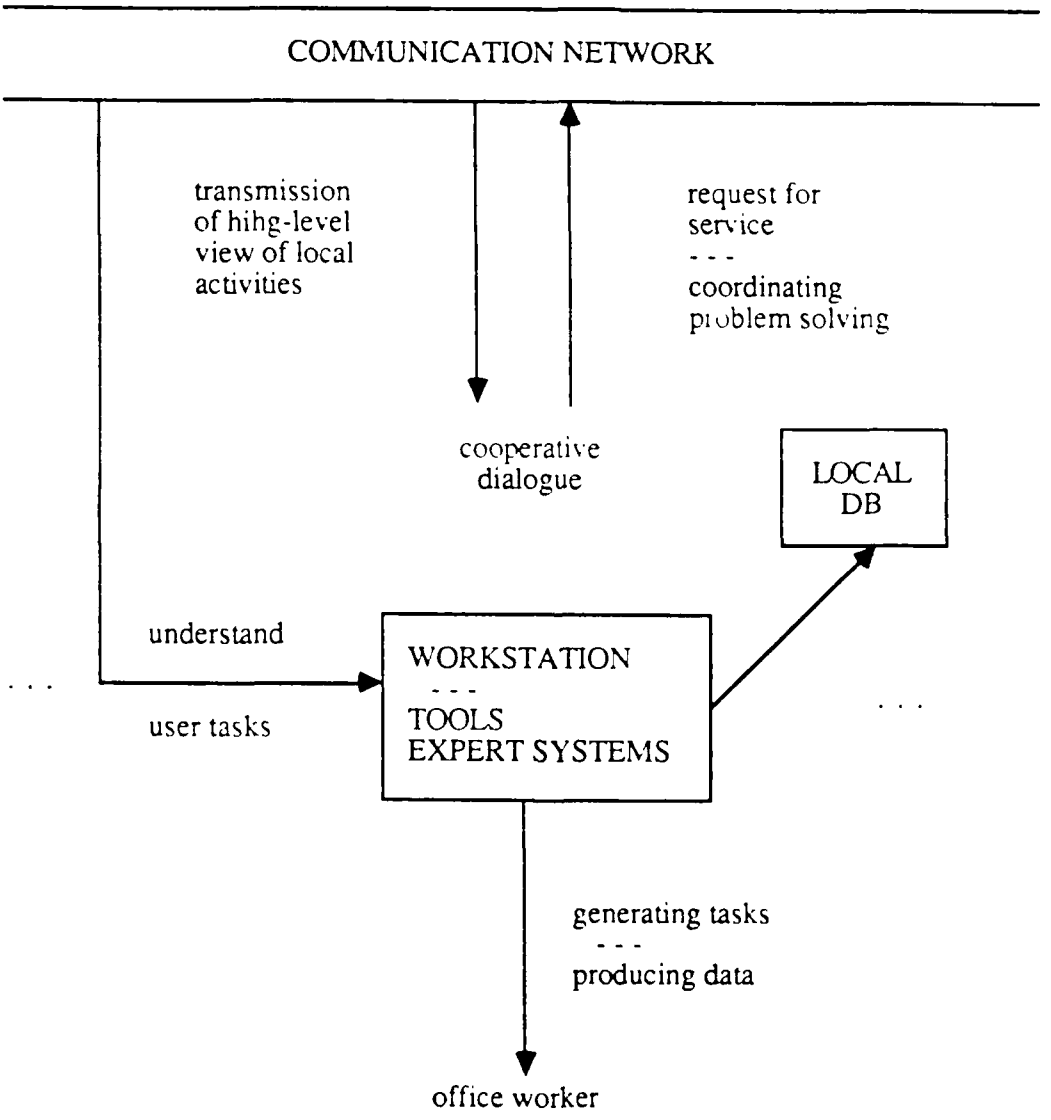


Figure 1. A node in a network of cooperative office workstations.

To illustrate the problem of local scheduling that takes into account global coherence, consider an office consisting of an executive, E, and his/her secretary, S. Suppose E is dictating letters to S, and the telephone rings. S answers, and the call appears to be about a very important shipment, and S is asked to provide some information about it. The scheduling choice here is between continuing with the letters (task T1) and performing the request that came over the phone (task T2). We want our system to consider a number of factors here, including the relative importance of the tasks (say, a number of people may be idle in the company because of the lack of raw materials that are to be shipped), the time limitations (suppose, the information is needed before the end of the business day, and it's already 4 p.m.; also, the estimated time of finding the requested information), personal characteristics of S and E, etc. If the secretary were scheduling purely locally, he/she may prefer to schedule T2, but knowing that E will be detained by her doing so, S may prefer T1 based on global coherence considerations. S's knowledge about personal characteristics of E can also be a factor: if E is very conscious of his/her status and importance, then the decision of scheduling T1 is even more strengthened; if not, and if S has the characteristic of being assertive, T2 may be preferred, after an explanation to E.

In what follows we, first, trace the project's genesis from three research projects in connected fields and discuss its functionality. Second, we describe how an office can be modelled in a distributed computer system such as OFFICE and describe its architecture and the basic processing cycle. Finally, we give an example of OFFICE operation where we concentrate on its reasoning capabilities.

The Task-Oriented Perspective.

Our initial effort in developing an expert system in the office domain is the task support system POISE (Croft et al., 1983). POISE has been designed to support office workers in their problem solving activities through the use of plan recognition and planning. In the plan recognition mode the system obtains messages about certain atomic events (such as tool invocations) and tries to determine into which of typical tasks known to the system this event fits. In this manner POISE is able to monitor the activities in an office, predict future activity and detect errors. If, as a result of the monitoring, the system understands the user's task, it can in principle take over its completion. This task completion mode is integrated with the planning mode of operation. In the planning mode POISE is supplied with a typical tasks and its parameters and tries to execute as much of it as possible, based on its knowledge of the task structure and the status of domain objects in a semantic database.

POISE's knowledge takes the form of an hierarchy of typical tasks. Each task is represented by a precondition statement that defines the necessary conditions for its execution; a goal statement that specifies the intended effect of the task; the sequence of subtasks needed to be performed in order to accomplish the task and the constraints among the parameters of the subtasks and those of the task. See Figure 2 for an example.

```
PROC    Purchase-items (Purchasing Amount Items Vendor
DESC    Procedure for purchasing items with non-state funds
IS      Receive-purchase-request
        ! (Process-purchase-order | Process-purchase-requisition)
        ! Complete-purchase

COND
        Process-purchase-order Amount          = Receive-purchase-request Amount
        OR
        Process-purchase-requisition Amount    = Receive-purchase-request Amount
        Process-purchase-order Items           = Receive-purchase-request Items
        OR
        Process-purchase-requisition Items     = Receive-purchase-request Items
        Process-purchase-order Vendor          = Receive-purchase-request Vendor
        OR
        Process-purchase-requisition Vendor    = Receive-purchase-request Vendor
        Process-purchase-order Amount          = Complete-purchase Amount
        OR
        Process-purchase-requisition Amount    = Complete-purchase Amount
        Process-purchase-order Items           = Complete-purchase Items
        OR
        Process-purchase-requisition Items     = Complete-purchase Items
        Process-purchase-order Vendor          = Complete-purchase Vendor
        OR
        Process-purchase-requisition Vendor    = Complete-purchase Vendor

WITH    Purchaser  = Receive-purchase-request Purchaser
        Amount     = Receive-purchase-request Amount
        Items      = Receive-purchase-request Items
        Vendor     = Receive-purchase-request Vendor
```

Figure 2. A plan in POISE

POISE plans are structured so that they in principle allow concurrent execution of subtasks of a task. Straightforward transformation of POISE into a distributed system cannot, however, be performed. Since POISE does not have a developed agent-oriented perspective, there is no way in it to express a fact such as 'requests made by the manager of the office have priority over those made by other workers' or the fact that even though certain workers are better at doing certain types of jobs, if they are not available to do a job of this type, then other workers have to be assigned this responsibility. There is also no way of talking about seemingly independent tasks being actually parts of a cooperative problem solving situation. This includes the considerations of arbitration of competing claims for limited resources.

POISE does not distinguish or reason about the agents' roles and the objects in plans. Thus, for instance, it does not have the possibility to understand that an unusual event happened if it gets the message that the president of a company typed a letter (and not a secretary). Therefore it cannot infer that the secretary may have a day off or that a goal must be instantiated of changing workload distribution among the employees.

Another deficiency of POISE is that the plan recognition and planning architectures are not designed for being distributed and assume a global blackboard and a single locus of control. POISE gives us some ideas about what an intelligent assistant could be but its architecture is not appropriate for use in a distributed environment and it lacks a distributed agent-oriented perspective.

The Distributed Agent-Oriented Perspective

One of the research areas where we can look for ideas of how to implement the distributed agent-oriented perspective is the field of distributed AI. One of the current approaches there is the study of functionally accurate, cooperative (FA C) distributed problem solving (Lesser and Corkill, 1983; Corkill, 1982; Durfee et al., 1984, 1985). With this approach, a problem is solved in cooperation by a set of semi-autonomous processing nodes (agents) that may have inconsistent and incomplete local databases, each node independently generates tentative partial solutions, communicates them through a network to other nodes, receives messages (partial solutions, goals, plans and facts) from other nodes, and modifies its processing in accordance with new input. The experience of this work has shown that the control problem is difficult; that the network communication is both difficult and computationally expensive; most importantly, it was found that the key to global coherence is having sophisticated agents who can reason about their own view of processing as well as the views of other agents. They have developed a system in which each node is guided by a high-level strategic plan for cooperation among the nodes in the network. This plan, which is a form of metalevel control, is represented as a network organizational structure that specifies in a general way the information and control relationships among the nodes. Examples of this information include static priorities among local tasks, to whom and what information to communicate and how to prioritize tasks that have been requested by other nodes versus those that were locally generated.

Other work by Smith and Davis (1981) has focused on the knowledge and the protocols necessary for nodes to decide in a distributed way how to allocate subtasks to other nodes. This involves a two-way bidding protocol in which the contractors (taking on the task perspective) and bidders (taking on an agent perspective) communicate to determine the best task allocation.

The work by Lesser et al. focuses on how to do local scheduling given a static task allocation that may redundantly allocate tasks among nodes, while Smith and Davis focus on dynamic task allocation. The office domain requires an integration of both approaches together with augmenting the knowledge used by both approaches for scheduling. The office domain also presents challenges to both approaches because of the tighter and more complex interactions among agents that exists in this domain, compared to the distributed interpretation domain from which both of the above approaches evolved.

The Cognition-Oriented Perspective.

The distributed problem solving approaches described above concentrated on the architecture of the network and the nodes, with the view of organizing the control structure. The types of knowledge necessary for control and communication in OFFICE are studied in the field of cognitive agency research (e.g. Georgeff, 1984, Moore, 1985, but mainly Nirenburg et al., 1985, 1986). The view of the world in this field is that cognitive agents are immersed in a world which is non-monotonic, in the sense that changes in the world can be introduced not only because of the activities of a single agent but also through uncontrolled external events. Agents are capable of a variety of cognitive tasks. They can perceive objects and events in the world. They possess a set of goal types and means of achieving goals of these types: plans. They perform goal and plan generation, selection and execution in complex situations in which many goals and plans coexist and compete for the attention of the agent's conscious processor.

The study of the knowledge that underlies the reasons for particular choices of goals and plans by an agent (in other words, reasons for scheduling and communication decisions) is the central theme of this approach. This knowledge is claimed to involve such factors as personality traits, and physical and mental states of the agent, in addition to the knowledge about the domain situation and the typical tasks and goals. Our approach is to use all the types of knowledge discussed in the cognitive agency approach within the architectural framework inspired by the distributed AI research.

2. An Architecture for a Distributed Office System.

We present here, through an example, an architecture for an intelligent assistance system that integrates the task-, agent- and cognition-oriented perspectives.

2.1. Representing an office.

An office is modelled as a network whose nodes are interpreted as office workers and edges, as communication channels. Every node in the network is a complete problem solver that consists of an office worker and his/her workstation. Following POISE, OFFICE deals with typical activities in a university-based research project (RP), namely purchasing equipment, hiring and travel. The types of agents in the RP office include Principal Investigator (PI), Research Associate (RA), Graduate Student (GS), Secretary (S), Vendor (V) and Accountant (A). A typical instance of a project may involve 1 PI, 2 RA's, 6 GS's, 1 S, 3 V's (e.g., DEC, Symbolics and TI) and and 2 A's (say, one in Accounts Receivable and one in Personnel).

Figure 3 shows the communication channels for the RP office.

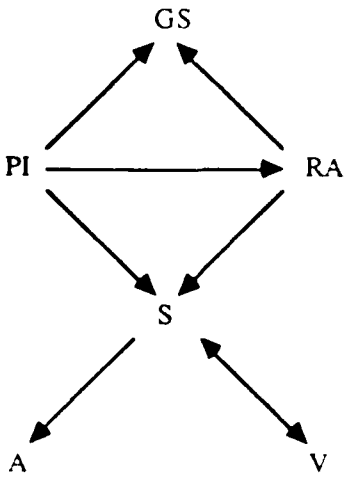


Figure 3. The network of processing nodes in a model of an RP office.
The arrows illustrate authority relationships (see below).

Every node in the office network is aware of its responsibilities to carry out parts of certain plans. They also know who or where from they can and should seek information that is necessary for them to perform their tasks (recall that information about the typical agents for all types of tasks is among the knowledge that every agent possesses).

At any moment t each agent in OFFICE has an agenda of current goals or, more precisely, of current goal instances, as illustrated in (1),

$$\left\{PU^{5}_{\sigma_1}, PU^{6}_{\sigma_1}, HI^{1}_{\sigma_3}, TR^{2}_{\sigma_4}\right\}_t \tag{1}$$

where PU^i, HI^j and TR^k stand for instances of goal types *Purchase*, *Hire* and *Travel*, and σ_i designate subsets of network nodes that are working cooperatively on particular goals. Intuitively, at any given moment the office workers are pursuing a number of goals, working in teams. Note that some such goals can be in conflict or can compete for resources. Therefore, the agents must have means of resolving these conflicts.

The architecture of an agent in OFFICE is illustrated in Figure 4. A frame-based representation is used for objects, goals, plans and actions, including messages. Plans are represented in extended EDL (cf. Nirenburg et al., 1985). An agent has knowledge about the goals it is typically responsible for as well as about plans that are typically used to accomplish these goals. (If node A has a goal G on its agenda, then A is *responsible* for achieving G.) It also has the knowledge about the current state of its goal agenda, as well as a subset of the contents of other agents' agendas. Scheduling knowledge used by the agent to select goals and plans for processing is represented as a set of condition-action rules. The agent also is aware of the authority relationships in the office, illustrated in Figure 3, that are part of the agent's scheduling knowledge.

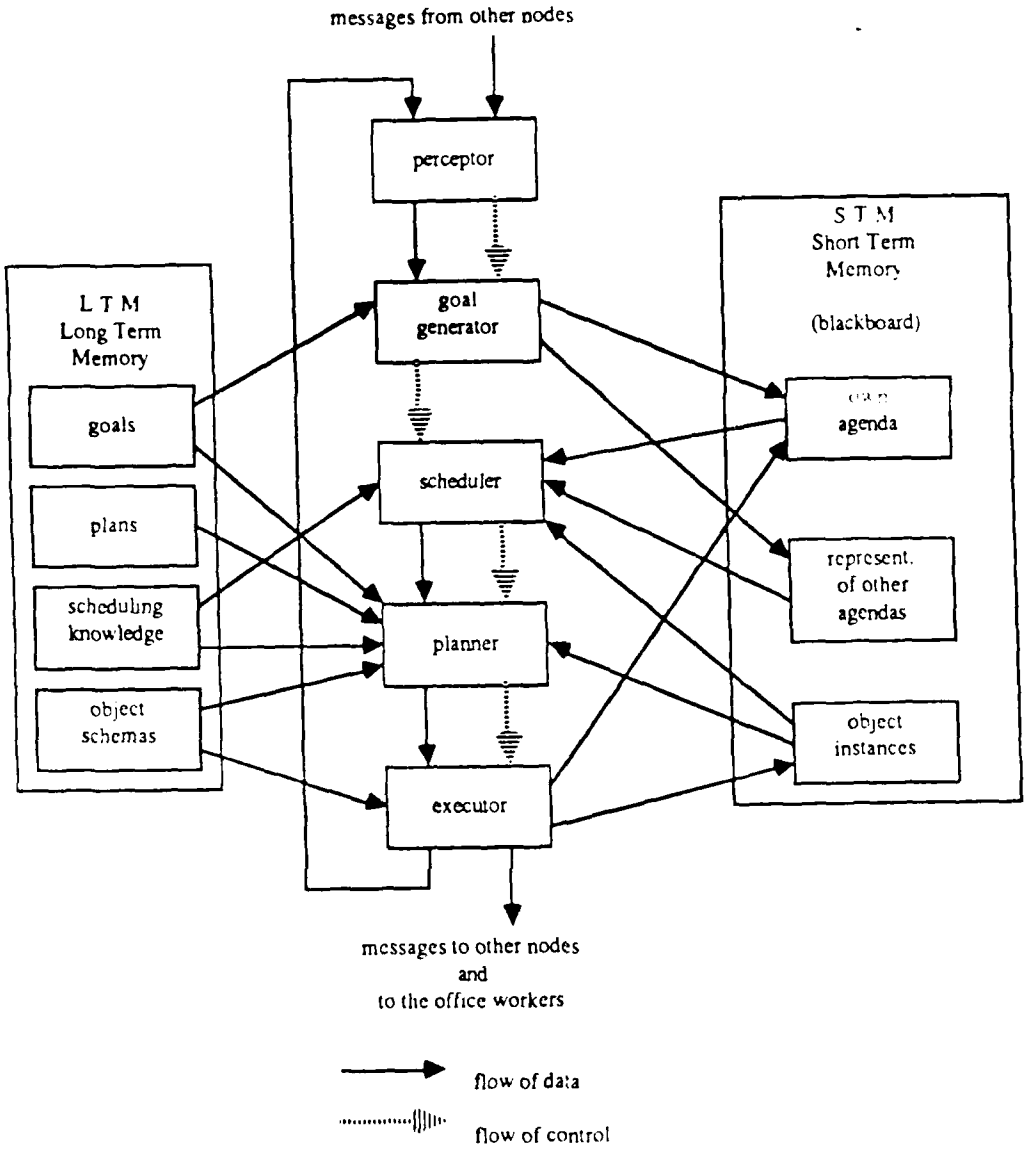


Figure 4. Node Architecture

Representations of two top-level goal types in OFFICE are given in Figure 5. A number of OFFICE plans are illustrated in Figure 6.

```
(goal HIRE-PERSON
  (Typical-Responsible-Agents RP PI)
  (Typical-Plan Hire-plan)
  (time-scale days)
  (importance 2)
```

```
  (beneficiary Research-Project)
  (Supergoals  Conformity-between-Workers-and-Work-Amount
               Use-All-Resources-Available)
  (Trigger (or (sum of expenditures is less than available funds)
               (there are less workers than needed to do work)))
)

(goal PURCHASE
  (Typical-Responsible-Agents  PI)
  (Typical-Plan Purchase-plan)
  (time-scale days)
  (importance 1)
  (beneficiary (PI S RA GS))                    ;any member of RP
  (Supergoals  Get-Equipment
               Use-All-Resources-Available)
  (Trigger  (and (there are funds available)
                 (the beneficiary's resources are incomplete
                  compared to the typical resources allocated
                  to this role-holder)))
)
```

Figure 5 The goals HIRE-PERSON and PURCHASE

```
(Purchase-plan
   (icon PU
   (With ((Agent RP member)
          (Object POBJ)  ;is not specified at the moment of
                         ;plan instantiation
          (Amount int)  ; - "-
        (      ))
   (is ((specify-item-to-buy (agent = RP member
                              object = item
                              approx-price = int))
        (make-document (agent = RP member
                        doc-type = purchase-request
                        object = item))
        (communicate (agent = RP member
                      destination = Secretary
                      object = purshase-request))
        (plan-selector ((process-purchase-order (agent = Secretary
                                                 object = item))
                        (process-purchase-requisition (agent = Secretary
                                                       object = item)))
            both plans are compound
        (complete-purchase (agent = Secretary
                            object = item))
)
   (preconditions  (Agent has money  Vendor has Object))
   (effects (Agent has less money
             Vendor has more money
             Agent has Object))
  )
```

```
(Process-Purchase-Order
   (with (agent = secretary
          object = item
          destination = vendor
          price = int))
   (preconditions (approx-price < $250 ))
   (is (make-document (agent = secretary
                       doc-type = purchase-order
                       object = (item vendor)))
       (communicate (agent = secretary
                     object = purchase-order
                     destination = vendor)))
)

(Complete-Purchase
  (with (agent= secretary
         object = item
         source = vendor))
  (is (# (communicate (agent = vendor
                       destination = secretary
                       object = item))
         (communicate (agent = vendor
                       destination = secretary
                       object = bill)))
      (check-goods (agent = secretary
                    object = item))
      (plan-selector ((pay-for-goods (agent = secretary
                                      destination = vendor
                                      object = item
                                      amount = bill amount)
                      (cancel-goods  (agent = secretary
                                      destination = vendor
                                      object = item
                                      amount = bill amount))))
)

(make-document              ;a primitive plan
  (with (agent = person
         doc-type = purchase-request | bid-request | purchase-order |
                    disbursement-form | item-rejection-form | cv | offer
         destination = person | organization
         object = (item price )  parameters that are mentioned in
                                 the document
  )
  (is primitive)
  (effects      ) ;the document exists
)

(check-goods
  (with (agent = person
         object = item))
  (is primitive)
)
```

```
(Communicate
 (with (agent = person
      destination = person
      object = message
      type = assertion | question | order
      instrument = medium))
                ,medium is a list of phone, mail, csnet  etc
 (is primitive)
 (action-for-primitive))


(Get-Info
  (with (agent = person
      object = message
      was-invoked-by = person3
      instrument medium)
 (is (plan-selector (ask-track (agent = person1
                          destination = person2
                          was-invoked-by = person3
                          instrument = medium))
                 find-track (agent = person
                          object = message)))
 (effects (communicate (agent = person1
                  destination person3
                  object = message
                  instrument = medium)))
)
```

Figure 6. A sample of OFFICE plans.

Local and Global Scheduling Knowledge

A special part of the knowledge in OFFICE is the knowledge about scheduling and prioritizing activities by the nodes in the network. A part of the scheduling knowledge is static, that is, is considered true irrespective of the circumstances in which the scheduling takes place. The other portion of the scheduling knowledge is dynamic in that it takes into account the presence of other goals on the node's agenda and the suggests the ways of dealing with goal conflict.

The static part of an agent's scheduling knowledge includes the authority and responsibility structure of the office and the profiles of actual workers in specific roles within the organization. The latter includes both the workers' stated attitudes and preferences with respect to the types of jobs they are performing and their personality profiles, as understood by the current agent, on the basis of which the above attitudes and preferences can be inferred.

The dynamic part of this knowledge includes a snapshot of problem solving activities from the current agent's perspective, a representation of time and other resources; and a set of operational rules that contribute to the task of scheduling. In this paper we will present these rules as a set of scheduling heuristics, bypassing, for the sake of clarity and understandability the actual formalism in which they are expressed. The scheduling

heuristics are as follows:

1. Static priorities are stated for all the types of top-level goals. The instances of goals with higher static priorities will be preferred. Thus, for instance, *Purchasing* can be declared more important than *Hiring*.
2. The more time a goal spends on the agenda, the higher the priority it acquires.
3. The less time the accomplishment of a goal will take (as estimated by an agent), the higher the priority it acquires.
4. The smaller the effort needed for the accomplishment of a goal (as estimated by an agent), the higher the priority it acquires. This rule measures effort in terms of both the amount of energy exertion on the part of the agents and the number of intermediate steps (plans) still estimated as needed to accomplish the goal.
5. If a precondition for a plan selected to achieve a goal is false, the goal's priority goes down; however, for specific types of preconditions and nodes a new goal of satisfying this precondition can be established.
6. The higher the authority of the node responsible for a goal, the higher the priority it acquires.
7. Beliefs about the agendas of other network nodes weigh less in the decision process than the contents of own agenda. For example, if the level of authority responsible for a goal G is *inferred* by a node then it will increase the priority of G to a lesser degree than in the case when the authority level was explicitly obtained as input.
8. If the accomplishment of a goal satisfies preconditions for the execution of a plan (or a number of plans) leading to the achievement of other goals (on any of the goal agendas in the network), the priority of the goal is considered higher.

The influence of prioritizing rules based on the above scheduling heuristics is calibrated to produce a general dynamic priority for every goal on a node's agenda.

2.2. How Do the Agents Operate?

A cycle of processing by each agent involves a consecutive invocation of the perceptor, the goal generator, the scheduler, the planner and the executor (cf. Figure 4).

The **perceptor**

obtains as input (either through the network or from the office worker) *messages* about changes in the world that were received since the previous time cycle (changes are various new states, including results of actions performed by agents in the system).

Input messages are classified according to their *speech act* character. Messages can be either assertions or requests. Assertions can be definitions, opinions, facts, promises, threats and advice. Requests can be questions (request-info) or commands (request-action). Commands are orders, suggestions or pleas. This classification is needed to improve the understanding capabilities of the system (as compared, e.g., with POISE). Also, it allows a clear way of setting goals for the nodes in the network.

Next, the perceptor 'understands' these actions in terms of *plans* they are parts of and correspondingly, in terms of what was the *goal* that the agent of that action pursued. This step embodies the *plan recognition* activity of the system, since, in the general case, it must understand plans of others in order to perform its own plan production.

The **goal generator**

updates the agenda of the node's goals due to new inputs. Thus, the arrival of the following input:

```
(message-14
  (instance-of message)
  (speech-act order)
  (sender PI-1)
  (receiver Secretary-33)
  (proposition (communicate Secretary-33
                            Vendor-101
                            'what is the price of desk-22'
                            Phone))
```

will lead to the generation of the low-level goal instance 'Get-Info-34' that will be fulfilled when the secretary knows the price of the desk. The plan selection for reaching this goal also is specified in the message: using the telephone. 'Get-Info-34' is added to secretary's agenda of goals to accomplish.

There are thus two kinds of sources of goals for every node. One source is the state of the (office) world (if there are more workers than workstations, the goal of purchasing equipment will be generated and put on the office head's agenda). The other source, as in the above example, is messages (requests and orders) from other nodes.

The **scheduler**

selects a goal to pursue from among a number of candidate goals on the agenda. It applies condition-action rules designed on the basis of the above scheduling heuristics and evaluates the current local state of problem solving from the current agent's perspective. After the scheduler finishes operation, one goal from the node's agenda is selected for processing, and control is passed to the planner.

The **planner**

has the task of providing a plan for the achievement of the goal scheduled by the scheduler. If the agent knows of a *canned plan* that typically leads from the current state to the goal state, the planner simply passes the plan to the executor (see below). If more than one plan can be used to achieve a given goal, the planner selects one of them, based on the scheduling rules. The same heuristics that are used for scheduling goals are also used for plan selection. This is in itself a scheduling heuristic.

The knowledge needed by the planner includes the list of plan types, the list of plans that are believed by the node to be instrumental in achieving the goal selected by the scheduler, and the for competing plans.

The **executor**

is called after the planner selects a plan for achieving the current goal.[1] It performs the following sequence of steps:

a) creates an instance of the chosen plan (if such an instance does not already exist) and lists it under the corresponding goal on the agenda.

b) checks preconditions of the plan: if preconditions do not hold (the plan is not immediately applicable) **then sets** precondition states to be (sub)goal states; puts them on the goal agenda (note that one of preconditions is 'to have values for all non-optional parameters') else expands the agenda tree by substituting the current plan by the sequence of its component plans.

c) if the first subplan in this sequence has the current node as its agent, it is processed by the executor; if another role in the office is the agent of a subplan, the execution of the current plan is interrupted and a value of its 'status' slot is set to 'suspended' and a corresponding message is issued to the agent of the next subplan.

d) if the plan is 'primitive' the actions specified in it are performed. Then the executor checks whether the plan is completed; if yes, the executor reports this, through the communication channels, to the node responsible for supergoal of the goal which the current plan helped achieve. In this way responsibility relationships are both statically and dynamically introduced into the system.

3. An Example Run of OFFICE.

We will consider 2 top-level goals: Purchase and Hiring. The processing will be traced from the standpoint of one specific network node, that of Secretary (S). At the beginning of the run S already has a *nonempty* agenda of plans and goals. It also has a representation of agendas of other nodes in the network. This representation may contain mistakes, because it is mainly a result of plan understanding activities of the node. The contents of S's agenda and S's belief about the agendas of a sample of other nodes at the beginning of our manual trace are given in Figure 7.

[1] This is a simplification. In reality planning and execution steps can be interleaved.

S's own agenda

AGENDA ITEM 1
Purchase-plan3 (object = terminal)
  communicate (agent = S destination = PI, object =
    [**communicate** (**agent** = V1 object = terminal
      destination = S)
     communicate (**agent** = V1 object = bill
      destination = S)])
  check-goods (agent = PI object = terminal16)
  plan-selector (agent = S object =
    [**pay-for-goods** (agent = S destination = V1
      object = bill)
     cancel-goods (agent = S destination = V1
      object = (terminal16 bill))]

AGENDA ITEM 2
process-purchase-order5 (object = book)
  make-document (agent = S document-type = purchase-order
    object = book destination = V2)
  communicate (agent = S object = purchase-order destination = V2)

Secretary's beliefs about PI's agenda

AGENDA ITEM 1
Purchase-plan3 (object = terminal16)
  complete-purchase (agent = PI object = terminal16)

AGENDA ITEM 2
Hiring-plan2 (RA)
  evaluate (agent = PI object = candidate3)
  make-document (agent = S object = offer destination = candidates)
  communicate (agent = S object = offer destination = candidates)
  select (agent = candidate object = accept-rej)
  make-doc (agent = candidate object = accept rej)
  communicate (agent = candidate object = accept rej)
  plan-selector (agent = S object =
    acceptance-track rejection-track)

S's representation of RA1's agenda

AGENDA ITEM 1
PU1 (object = book11)
  process-purchase-order (agent = S object = book11)
  complete-purchase (agent = S object = book11)

An agenda item consists of the name of a goal and the names of those of the plans selected for its accomplishment that are not yet (completely) executed, with the bindings for their parameters. Plan names are printed in **bold**. Plan names with numbers appended represent plan instances. The above agendas say that the secretary has the plans to facilitate the purchase of a terminal and to facilitate purchasing of a book asked for by a research associate (Purchase-plan1); S believes PI has plans to hire a research associate (Hiring-plan2) and to facilitate the purchase of a terminal (Purchase-plan3). S also believes that RA1 has the plan of purchasing a book (Purchase-plan1). PI is responsible for both g on its agenda; S is co-responsible for the Purchase-plan3. In contrast, S is responsible only for a subplan of the top-level plan Purchase-plan1. RA1 is responsible for Purchase-plan1.

Figure 7. Sample Contents of the Agendas of an Agent.

Now let us trace the operation of OFFICE through a number of time slices starting with the above state, observing the decision S makes and the changes to its agenda due to new inputs.

------ time slice 1 ------

Suppose, there is a message posted on the secretary S's blackboard : message19 from research associate RA2, of type *order*, that asks to get a price for a desk from vendor V by phone. This message is received by S and a new goal, GET-INFO11, is generated and put on its agenda. S also updates its representation of RA2's agenda by adding there the (inferred) plan of buying a desk. Note that the inferred Purchasing goal is not on S's agenda; therefore, S is not responsible for it.

Next, the scheduler must choose one of the 3 goals on the agenda (PU3 P-P-O5 and GET-INFO11) for immediate processing.

In our example the Get-Information goal will be chosen. This happens because the Purchasing goal is out of contention since it is in the stage of waiting for ordered goods (terminal) to come (Scheduling Heuristic 5). The choice is, therefore, between the Process-Purchase-Order and the Get-Information. P-P-O has, of course, been on agenda for a longer time (Scheduling Heuristic 2), but GET-INFO can be performed by just placing a phone call, while P-P-O requires typing out a form (Scheduling Heuristic 4). There is no rush on the book order, so the goal that can potentially be achieved sooner (Scheduling Heuristic 3) is selected (Scheduling Heuristics 2 and 3 prevail in this case over Scheduling Heuristic 4).

Next, a plan **get-info** is found for achieving the chosen goal; this plan is instantiated and the executor runs its first subplan: **communicate15** (agent = S, object = message34, proposition = message19.proposition, destination = V2, type = question, instrument = phone). As a result of that subplan, the vendor is informed about the

question.

------ time slice 2 -----

New inputs: a) Message20: a terminal and a bill arrived from vendor V1 b) Message 21: the price for the book arrived from V2.

The messages are perceived and understood as the execution of specific plans traced on S's agenda: a) refers to the two *communicate* plans that are objects of the next component of the plan chosen for the Purchase3 goal instance; b) is the response to message19 above.

The above messages do not lead to the generation of any new goals. The scheduler now has the following choice: PU3, P-P-O5 and GET-INFO11. P-P-O5 has the same status as at the previous cycle. PU3 is now at a point where the PI must be told that preconditions are met for the execution of the *check-goods* plan (because the terminal arrived). Only one action remains to be performed in GET-INFO11, and that is to relay the information obtained from V2 to RA2.

At this point GET-INFO11 is chosen for the following reasons. S knows that PI is currently in a meeting with a candidate for hiring. Even though the importance of the *check-goods* plan is relatively high (Scheduling Heuristic 1), it cannot be performed at this point (the presence of PI is necessary) and is therefore rated low. GET-INFO11 is closer to completion than the other goals. In accordance with Scheduling Heuristic 3, it is selected, and S sends the plan (**communicate** agent= S, Destination = RA2, *Object = Message21.proposition) to the executor.*

After this plan is executed, the entire tree for GET-INFO11 is deleted from the agenda.

4. Summary and Status.

We hope we have shown that in order to provide assistance in distributed office environments we need to integrate the agent-centered, the task-centered and the cognition-oriented perspectives. It is important to carefully choose the task and delineate the world corresponding to it. It is equally important to provide an architecture that can support sophisticated scheduling activities by nodes in a distributed problem solving network. At the same time one should try to explore the sources of real-world knowledge that is used as the basis for scheduling. In addition to the observable world situation the scheduling algorithm must have access to the knowledge about the internal states of the processors, or, in other words, the 'personality profile' of the agents to whom the system provides assistance.

The node-level knowledge and processors have been implemented in Zetalisp on a Symbolics 3600 Lisp Machine. We are currently developing the network level of the system.

References

Corkill, D.D., 1982. A Framework for Organizational Self-Design in Distributed Problem Solving Networks. Ph.D. Dissertation, University of Massachusetts, Amherst. (Available as COINS Technical Report 82-33.)

Corkill, D.D. and V.R. Lesser, 1983. The use of meta-level control for coordination in a distributed problem solving network. Proceedings of 8th IJCAI, 748 - 756.

Croft, B.W., L.S.Lefkowitz, V.R. Lesser and K.E. Huff, 1983. POISE: an intelligent interface for profession-based systems. Proceedings of the Conference on Artificial Intelligence, Oakland University, Rochester, MI.

Durfee, E.H., D.D. Corkill and V.R. Lesser, 1984. Distributing a distributed problem solving network simulator. COINS Internal Memo, University of Massachusetts.

Durfee, E.H., D.D. Corkill and V.R. Lesser, 1985. Increasing coherence in a distributed problem solving network. Proceedings of Ninth IJCAI, Los Angeles, August 1985, 1025 - 1030.

Georgeff, M., 1984. A theory of action for multiagent planning. Proceedings of AAAI-84, 121 - 125.

Lesser, V.R., D.D.Corkill, 1981. Functionally accurate, cooperative distributive systems. *IEEE Transactions on Man, Systems and Cybernetics*, SMC-11, 81-96.

Lesser, V.R., D.D.Corkill, 1983. The distributive vehicle monitoring testbed: a tool for investigating distributed problem solving networks. *AI Magazine*, 4, 15-33.

Nirenburg, I., S. Nirenburg and J. Reynolds, 1985. POPLAR: Toward a Testbed for Cognitive Modelling. Technical Report COSC7, Colgate University.

Nirenburg, S., I.Nirenburg and J. Reynolds, 1986. Studying the Cognitive Agent. Technical Report COSC9, Colgate University.

Smith, R.G. and R. Davis, 1981. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11, 61-70.

POPLAR: A Testbed for Cognitive Modeling.

Irene Nirenburg, Sergei Nirenburg and James H. Reynolds

Colgate University, Hamilton, NY 13346

ABSTRACT

This paper presents an overview of a cognitive modeling system centered around a personality-oriented planner, and then describes in detail the types of knowledge it uses to make control decisions. POPLAR is a model of an intelligent actor capable of planning sequences of control and domain actions in a simulated world that exists independently of the planner. The world is a simplification of the 'Dungeon' computer game environment. The actor makes control decisions on the basis of situational knowledge as well as its personality characteristics (character traits, physical and mental states) and its beliefs about personality of other cognitive entities in the world. POPLAR is a step toward an AI system whose behavior is psychologically justified and can provide the basis for an experimental testbed in cognitive modeling.

1. Setting the Stage.

The POPLAR planner is a component in a model of an intelligent actor. It is an approximation of the human actor in that:

i) like humans, it possesses multiple goals with associated plans;

ii) like in humans, its control decisions depend upon multiple sources of information, e.g. input from the 'objective' world, its permanent character traits, its temporary physical and mental states, and past experience;

iii) like humans, it is immersed into an 'objective' world, changes in which can be introduced not only by the actor, but also by events beyond the actor's control, making it necessary to deal with non-monotonicity.

We believe that the essence of an intelligent actor's cognitive activity is best described in terms of the following loop:

1) perceive input stimuli (sensory, proprioceptive or mental);

2) generate goals connected with these stimuli;

3) schedule the most important goal instance for the given period of time: the one to which the actor's cognitive resources are allocated;

4) choose (occasionally, create) and

5) execute plans to achieve this goal, including performance of physical, verbal or mental actions that are components of these plans. Executions of the loop provide continuous change and stimulation at several levels. Physical actions introduce changes in the objective world. Verbal actions can provide sensory input for other intelligent actors in the world. Mental actions introduce changes in the world of the actor himself (his event memory and beliefs). So, the actions by the actor and other actors in the objective world change this world, and therefore, provide new inputs for the system.

POPLAR offers a solution to above loop components 2), 3), the non-creative part of 4), and the mental action part of 5). The visual perception portion of 1) and the physical actions of 5) are **simulated** through interaction with the human user of POPLAR.

In the current implementation there is no natural language capability (i.e. the verbal behavior of 1) and 5) are not addressed). Nor do we tackle in any complete and principled manner the extremely complex problem of learning (one facet of which is the creative part of 4).

The central cognitive and architectural points that distinguish the current version of POPLAR are, in addition to i) - iii) above, as follows:

A. The choice of the type(s) of knowledge for scheduling (cf. 3 above) and selecting (cf. 4 above) activities. We proceed from the assumption that in a non-trivial world these operations should be based on a psychologically justified model of human cognitive behavior. This property makes POPLAR personality-oriented, i.e. provision is made in the present model for introducing personality factors that influence goal generation and plan selection.

B. Decisions concerning the organization of metaknowledge that monitors and directs the cognitive processes of goal generation and plan selection. POPLAR represents such metaknowledge in the same framework as the domain plans (top-level, intermediate and primitive). This allows them to be processed by the same reasoning mechanism.

A discussion of POPLAR's relation to other work in the field is in Section 6.

2. The Conceptual Architecture of POPLAR.

The conceptual architecture of POPLAR, as presented in Figure 1, consists of the following modules:

1) the objective **world**, information from which and from
2) the **regulatory system** of the actor, where the non-cognitive knowledge about the actor's character and physical and mental states is stored (cf. Norman, 1981), is obtained by
3) the **sensor**, which processes this input and produces, in the **short-term memory** (STM) of an actor,
4) the **snapshot**, in which the objects currently perceived by the actor are stored, with their parameters, to be scanned by
5) the **goal generator** component of the reasoning mechanism (the **cognitive** module) which produces
6) the list of **candidate goals**, that contains all the goal instances that the actor has at a certain time, including the ones added after the new input was processed. In making its decisions, the **goal generator** uses the data stored in
7) the actor's **long-term memory** (LTM), which contains knowledge about
   a) the **beliefs** the actor has about
   - **objects** in the objective world, including self-beliefs
   - actor's **goals**
   - domain-specific and metalevel **processes** (stored as **plans**)

   b) the acquired **values** the actor has about these beliefs: what is more important, when and why, etc.

   c) the **event** memory that embodies past experience.

8) The **scheduler** component of the reasoning mechanism chooses (schedules) a goal instance in the list of candidates and selects the appropriate plan for its achievement. The **executor** component of the reasoning mechanism then attempts to execute the plan. Lower-level primitive plans are, in fact, actions that are performed by

9) the **output** module; these actions can introduce changes into the world, into the list of candidate goals and the long-term **memory**.

3. The Implementation.

POPLAR is an implementation of the above conceptual schema in a concrete application domain. It has been implemented in PEARL (Deering et al., 1981) which runs in Franz Lisp under Unix 4.2.

The world in which POPLAR is immersed is reminiscent of that of the well-known 'Adventure' or 'Dungeon' games. We represent a cave in which POPLAR's actor can find and react to enemies, treasures, tools, weapons, food and other objects. It is important to understand, however, that POPLAR is **not** a game-playing system. We are in the process of applying the system in a different domain (the office world).

At present POPLAR's actor is supplied with three basic goals:

1) 'Don't get killed', dubbed Preserve-Self-1 or PS1

2) 'Don't die of hunger, thirst or fatigue', Preserve-Self-2 or PS2

3) 'Collect as much treasure as possible', Get-Treasure or GTR.

In POPLAR the system is making the decisions about what to do next, while it is the responsibility of the user to provide it with input and means for verification of success of actions. The user, therefore, provides the testing ground for the system's empirical experience in the world.

With this caveat in mind, let us see how POPLAR is organized to allow its actor to 'act' in this environment.

4. The System Architecture of POPLAR.

POPLAR's system architecture (Figure 2) represents the conceptual architecture of Figure 1 with implementation restrictions superimposed.

In the current version of POPLAR the role of the objective world including the provision of its rules, 'the laws of nature', is assumed by the human user/experimenter. The user also interactively introduces and removes objects in the cave and modifies their parameters. (In future versions we intend to implement ongoing changes in the objective world generated by the operation of if-added demons on a World Blackboard (cf. 4.2.).)

The user also either permits or forbids certain primitive operations to simulate the actor's pragmatic experience. For example, the user might forbid the actor to pick up an object that is 'too heavy' but previously believed by the actor to be manipulable. This natural state of affairs underscores the difference between the objective world and the world of POPLAR's actor and his beliefs. It is also a means of modeling mistakes (a necessary first step in trying to learn how to recover from them).

The sensor and the output block are simulated in POPLAR's monitor (though mental actions are performed by demons (see below).

When the user decides to add an object to the current world, it does it by listing it on the **world blackboard** (WBB), the data structure interfacing the objective world and the world of POPLAR's actor. WBB also contains a clock which guides all temporally spread processing.

The STM of POPLAR's actor has the reasoning mechanism (the monitor and the executor with their associated bookkeeping functions, demons) permanently connected with it*.
STM contains one-instance metaplans: **the goal** generator and the scheduler. STM also includes the **actor blackboard** (ABB), which **contains** slots relating to the current state of POPLAR actor's activities, including notably the **agenda** of activated goal instances.

POPLAR actor's LTM contains his objects, plans, rating functions and history. Cf. a detailed discussion in 4.1.

POPLAR actor's knowledge about his own regulatory system and that of others is linked in the implementation with the representation of these objects in LTM. In addition to knowledge about objects, LTM contains knowledge about plans, history of processing and proper scheduling and selection.

Let us discuss the components of POPLAR in greater detail.

4.1. LTM.

4.1.1. Objects.

Several typical object frames and the semantics of their slots are described in Appendix 1. The choice of character traits is at present empirical. However, in parallel to implementing POPLAR, we have been conducting extensive psychological experiments seeking to establish the set of 'primitive' personality characteristics and their mapping into more complex notions that are used by intelligent actors in personality-based decision-making. A separate set of experiments will determine the primitives for specifying mental states of the actor.

4.1.2. Plans.

POPLAR's knowledge about the dynamics both in the objective world and in the actor world is represented as a set of declarative structures called plans.

Plans in POPLAR are classified into several groups (cf. Figure 3).

First, there are domain plans that describe actions in the world and metaplans that describe the processes that manipulate other plans. These include such plans as the goal-generator (gg), the plan-selector, the agenda-scheduler (as), etc. Second, there are top-level plans whose instances appear in POPLAR's agenda as representatives of the three main goals, and primitive plans that are no further decomposable into sequences of actions and provide the proper framework (of preconditions, effects, etc.) for their main action.

The plans that are neither top-level nor primitive are called intermediate. Intermediate plans are never scheduled other than in the process of expanding a top-level plan. There are no intermediate metaplans. Also, all of the metaplans are primitive (decomposable), and two of them, at the same time, top-level.

* The monitor, the executor and the bookkeeping functions stand out among the components of STM in that they are not 'conscious' functions: the actor performs them 'instinctively', while of other elements of STM the actor is consciously aware.

To illustrate the above discussion, consider, for instance the top-level plan of dealing with enemies, such as, in the POPLAR objective world, snakes, crocodiles or trolls. The actor can have a number of (intermediate plan) possibilities: to fight, to flee, to hide, to wait and see what happens, etc. All of the above are decomposed into strings of lower level plans (such as get, take, find, etc.), and the process of decomposition continues until all the final decopositions contain only primitive plans (such as, for instance, move or take).

Plans in POPLAR are represented in a modified version of the language EDL (Bates et al., 1981; cf. also Croft & Lefkowitz, 1984). The frame for a plan contains the following slots (clauses):

ID	the name of the plan
TOP-LEVEL-FLAG	is this plan top-level?
IS	contains the temporal and causal expansion of the plan
COND	used to pass parameters ('propagate constraints') to lower-level plans upward propagation will be added for the plan recognition task
WITH	specifies the parameters with which the current plan will be processed
CONTROL	contains predicates to choose whether to execute optional steps in the plan this slot has the form of an a-list: (<(Control# <s-expr>)>*
PRECONDITIONS	predicates that allow the processing of the current plan to start; differ in principle from CONTROL predicates by being independent of the current context of plan processing
STATUS	one of 'on-agenda', 'executed', 'succeeded', or 'failed'; used for communications with the reasoning mechanism
ACTION-FOR-PRIMITIVE	if plan is domain primitive permission is requested for its completion and the main action is performed (the rest being 'effects')
TIME	number of time cycles the plan takes (only for primitives) -- either integer or s-expression that evaluates to integer
RATING-FUNCTION	scheduling knowledge, see below
EFFECTS	auxiliary (including bookkeeping) modifications accompanying the success of the plan

Figure 4 contains a grammar of the plans implemented in this version of POPLAR, and Appendix 2 contains annotated examples of POPLAR plans.

4.1.3. The Rating Functions.

The knowledge that POPLAR's actor has about the relative importance of a top-level goal instance and the relative merits of one plan of action aimed at achieving a goal over another is embodied in the **rating functions**. In the current implementation rating functions are associated with every plan that can serve as parameters in the plan-selector and the agenda-scheduler.

The rating functions calculate a numerical value for a plan, a rating, in all situations where a choice among plans that can be pursued is possible. They draw upon:

a) knowledge of the objects involved in an objective world situation;

b) the character traits, mental and physical states of the actor;

c) the actor's beliefs about the character and current physical/mental state of any other cognitive entity participating in the situation;

d) the actor's event memory, the history of past processing.

Thus, if two actors, Actor1 and Actor2 find themselves in an identical threatening situation (e.g. a snake), but one of them is more courageous (a character trait) and or is in general not very fearful of snakes (a situational characteristic), the actors may respond to the situation by choosing different plans (e.g. Flee for Actor1 and Fight for Actor2) or even altogether different goals (while Actor1 is likely to choose 'Preserve-Self' against the snake -- because high levels of attention to threats can be expected from actors with low courage values; Actor2 may choose, say, an instance of 'Get-Treasure', because the snake is not serious enough a threat).

The construction of rating functions is an empirical process of gradual refinement. Even without changing the knowledge used by the rating functions one can always manipulate parameters of a function to calibrate its results.

One of the objectives of the psychological experimentation conducted in parallel with this project (cf. Section 7) is to better understand the nature and parameters of the rating functions.

Examples of rating functions are presented in Appendix 3.

4.1.4. History

This part of the actor's LTM contains his memory of past processing. In principle, history can have a very rich structure and be used in a wide variety of ways. Special demon-type functions can be defined, for example, to introduce modifications into the actor's beliefs about objects and processes in the real world based on certain patterns in the event memory[+]. This is one more location in POPLAR's architecture where a measure of learning can and is planned eventually to be introduced.

At present the history contains only two types of data: a) the record of all the recursive calls to the executor in the form of paths that the processing took in the grammar of plans and b) a list of the objects (physical or mental) found by all instances of the Find plan, this knowledge is used to retrieve the status and the results of various plan instances. A typical instance of history is presented in Appendix 4.

[+] An example: Suppose that in an internalized plan for fighting crocodiles 'stick' is listed as the best weapon. Then during one invocation of the plan Fight (Actor Crocodile Weapon) no stick could be found, so that Actor had to use a gun. It appeared that both the results were better and the fatigue increase was smaller. After this plan execution was written into the history, a comparison is made (by the above demon) and the belief about the stick being the best weapon is changed.

4.2. Actor/World Interfaces: the blackboard.

As mentioned above, in the current implementation of POPLAR there are two blackboards that facilitate links between the world and the actor.

4.2.1. The World Blackboard.

WBB is used for introducing new sensory input and managing temporal relations in the system. POPLAR has time-triggered demons that automatically update the values of the actor's physical and mental state based on the amount of time he engages in a certain activity.

In their simplest manifestations, the time-related modifications deal with increasing the actor's hunger, thirst and fatigue values at predetermined independent rates. When the value of any of the above parameters becomes greater than a predefined threshold, a message to this effect automatically registers in ABB's 'states-perceived' slot, as a result of which at the next pass of the monitor an instance of Preserve-Self-2 goal will be activated, and the corresponding top-level plan will appear on the agenda.

Temporal knowledge is also used to implement a simple model of attention. A detailed discussion of this mechanism will be deferred till Section 5.

4.2.2. The Actor's Blackboard.

ABB contains information about

a) the list ('objects-perceived') of object instances that the actor has perceived in the current environment;

b) the list ('states-perceived') of all physical states currently perceived that warrant the attention of the goal generator (e.g. the level of hunger above a threshold);

c) the agenda of all top-level plans (the representatives of the main goals) vying for the attention of the cognitive processor of the actor at any given time;

d) the stack ('current-path') of plans currently being executed (from a top-level plan to a primitive).

In future implementations, specifically when plan recognition will be added to the repretoire of POPLAR and the number of actors inhabiting its world will be allowed to be greater than one, the number of ABBs in the system may grow to as many as the square of the number of actors. This is because every actor stores his beliefs about other actors' activities in instances of ABB attached to his representation of these other actors. Therefore, each actor theoretically can be aware of all the other actors and contain an ABB for each, including himself.

A typical example of ABB and WBB contents is presented in Appendix 5.

5. The Algorithms.

5.1. The Monitor.

The top-level control function of POPLAR, the monitor, is an infinite loop (our actors do not die -- only if killed by enemies!) which performs the following tasks:

a) it maintains contact with the user (to obtain new input);

b) it starts the executor loop that consists of i) processing new input, ii) scheduling an action and iii) executing this action.

c) it displays selected situations in the world with the help of a (rather simple) graphic interface.

5.2. The Executor.

The main bulk of POPLAR processing is performed by the executor. To understand how POPLAR works it is sufficient to trace a cycle of its activities.

The executor is called many times during one monitor cycle. First, it processes the goal generating plans using the information obtained by the monitor from the objective world as well as that from the actor's regulatory system. As a result of this stage, the agenda of competing top level plan instances is updated. Second, it executes the agenda scheduler plan select the best candidate plan. Finally, it executes the chosen top-level plan (this involves a number of recursive calls to the executor). When eventually the execution ends, the result of current processing (success or failure) is reported, and a new cycle of the monitor begins.

Omitting a few overly technical details, we can describe the activities of the executor generally as follows:

a) obtain a plan to process; if it is not a plan **instance** (the agenda holds only plan instances, e.g. 'GTR19'; whereas **is** clauses of plans are formulated in terms of plan **types**, e.g. 'Find'), create a new instance of this plan;

b) check the plan's **preconditions** clause: if preconditions do not hold, report failure and its reason and exit; otherwise,

c) expand the plan by considering its **is** clause: call the **is** clause parser;

   c') if the **is** clause is 'primitive', then **action-for-primitive** is performed (most often this is a request to the 'laws of nature', the user, to allow an update in the objective world, e.g. a move by the actor; if the permission is given the processing proceeds as specified in e') below; if the action is not allowed the processing proceeds as in e''). (Let us repeat that the semantics of this situation is that the actor's beliefs about the objects and or plans and or values are somewhere wrong, as a result of which some indication of imminent failure must be given to prevent the 'automatic' success of most planners in situations where the internalized preconditions of a plan hold.)

   c'') if the IS clause is not 'primitive' the parser has to make specific control decisions: i) whether to execute an optional subpath in the IS clause; ii) which of any possible number of disjoined subplans to choose for fulfilling the current plan. (The ability to choose one of a number of 'shuffled' subplans (those that can be fulfilled in any temporal order) will be added to POPLAR in near future.) The knowledge about whether to execute an optional subpath is encoded in the **control** slot of the plan whose **is** slot is parsed. The knowledge selecting one of disjoined subplans is contained in the plan-selector metaplan and the **rating function** slot of the current plan. Once it becomes clear what member of the **is** clause should be processed first, the executor

d) calls itself recursively with this plan; this event is recorded on ABB (cf. 4.2.2), specifically, in a data structure called **current-path**; the old content of **current-path** is added to **history** (cf. 4.1.4.).

e') if an **is** clause is processed to its end (cf. the special case of 'primitive' in c') above), the **status** slot of the plan is set to 'succeeded' and the **effects** clause is evaluated;

e'') if for some reason the **is** clause cannot be processed to its end, the **status** slot is set to 'failed' and

f) this information is communicated to the parent plan, the current plan is discarded from the **current-path** stack, and the processing of the **is** clause of the parent resumes. When eventually, the outcome of the top-level (and bottom-of-stack in **current-path**) plan becomes known, then

g') if it succeeded, then the **effects** clause is evaluated and the corresponding top-level plan instance is removed from the **agenda** (and added to **history**);

g'') but if it failed, then, **assuming that** the need that had spawned this goal has not been satisfied, the executor **creates a new** instance of the same top-level plan and adds it to the agenda instead of the failed one (which goes to **history**)

h) a new cycle of the monitor starts

5.3. Modeling Attention.

The previous section described the normal flow of control in a monitor cycle. In real life, however, an actor can hardly have the luxury of being able to finish the processing of a top-level plan without taking in new information about the objective world. In future implementations of POPLAR the temporal relations among plans will be elaborated to include the many possibilities of concurrent processing (cf. Allen, 1983a, for the description of a model of time that can be adapted for use in our model; cf. also McCue & Lesser, 1983) for a temporal logic in the POLS system).

At present, however, POPLAR reacts to this problem as follows. When a top-level domain plan is chosen from the agenda and passed over to the executor, its rating is used for calculating the number of time cycles this plan will be allowed to execute without being interrupted. The more 'important' the plan (i.e., the higher its rating) the longer it is allowed to execute uninterrupted. This current programming device is a rough simulation of the actor's concentration or attention to the task. Intuitively, the more immersed one is into a task, the less one would be inclined to be distracted by new sensory inputs. It is obvious that character traits and physical mental states affect the ability to concentrate.

When an interrupt occurs, the entire **current-path** is suspended; the instance of the top-level plan is deleted from the agenda and another instance is created and added to it (the new instance reflects the knowledge of the stage at which the processing was suspended; **history** is used for this purpose). Then the monitor starts a new cycle.

5.4. An Example.

Suppose we want to test POPLAR's performance in the following situation of the world. We want to put the actor in a cave with a rock, a snake and an apple and to set its hunger well above the detecting threshold.

POPLAR acts as follows:

a) asks the user whether he wants to remove certain objects from the world, we do not, so we answer in the negative;

b) asks the user whether he wants to change any of the properties of the objects already present in the world; this is the time to input the (high) value of actor's hunger;

c) asks whether the user wants to add new objects to the world; we do; since our perception module is simulated, we submit prefabricated instances of objects to POPLAR, we write (rock1 snake1 apple1)

d) adds the above object instances to ABB.objects-perceived. Since snakes spawn the need for protection (by virtue of their being descendants of 'creature'), the goal Preserve-Self-1 is activated (by the gg-input plan) and an instance of its corresponding top-level plan, PS10, is added to ABB.agenda (which already contain the unique instance of the Agenda-Scheduler plan that resides there permanently); appropriate messages are issued by POPLAR;

e) detects, through **gg-states-perceived**, the actor's hunger; 'hunger' is added to ABB.states-perceived and an instance, **PS20**, of the top-level plan of the Preserve-Self-2 goal is added to ABB.agenda; appropriate messages are issued;

f) since no objects had been present in the world before, and, therefore, no changes to their properties could be introduced, gg-objects-perceived will not be needed in this case, a message to which effect will be issued;

g) at this point ABB.agenda is (agenda-scheduler PS10 PS20); the monitor calls the executor with the scheduler plan, as a result of which the two domain plans receive ratings. Suppose now that PS20's rating is higher (because the actor is very hungry and at the same time not too afraid of snakes); this being the goal choice,

h) the scheduler is called with PS20(Actor hunger); checks its preconditions (empty!) and expands its **is** clause: the plan-selector, using the rating functions in the plans Eat, Drink and Sleep, decides to select Eat; an instance of Eat, Eat0(Actor) is created and pushed onto **current-path**

i) Eat0's preconditions are checked (empty!), and its own **is** clause is expanded; this means creating a new instance of Find, Find0(Actor food Actor.inventory), -- that is, first the actor wants to check whether he is carrying some food;

j) *the control predicate chooses* whether to execute the optional Find and Get plans; the predicate essentially returns 'true' if the previous Find failed; the optional subpath corresponds intuitively to the situation when the actor looks around him trying to find some food; suppose now that Find0 fails; in this case,

k) Find(Actor food ABB.objects-perceived) is executed; Find's IS clause is 'primitive'; its **action-for-primitive** is to record the object found; Find1 finds apple1;

l) next, GET0(Actor, apple1) is created and pushed onto **current-path**; this instance's **is** clause consists of Move followed by Take; (in reality, Get has three parameters, the third being the indication of the time that the actor can spend on retrieving the object -- this is very handy as a precondition if, for example, an adversary can reach the desired object first!)

m) Move0(Actor Apple1) is created and pushed onto **current-path**; Move is a primitive plan, so its **action-for-primitive** asks the user for permission for the actor to move to the point where apple1 is. We grant the permission; Move0 evaluates its **effects**, updating the positions of the actor and all the objects in his inventory and sets its **status** to 'succeeded';

n) **current-path** is appended to **history**; Move0 is popped, and the next plan in the IS clause of Get0 is pushed onto **current-path**: Take0(Actor apple1);

o) Take0 is primitive; its processing is similar to the processing of Move0; it succeeds, one of its effects being that apple1 is added to the actor's inventory, and after manipulations with **current-path** similar to those in l), Ingest0(Actor apple1) is sent to the executor;

p) Ingest is primitive; suppose we allow the actor to ingest the apple; then, after the appropriate (and by now familiar) bookkeeping operations, we find ourselves at the point where

Eat0 is proclaimed as succeeded; at this point we evaluate its **effects** and pop it from **current-path** (which at the time contains only PS20, known to have succeeded)

q) **effects** of PS20 are evaluated (the hunger level of the actor is decreased, and a message to this effect is issued), and with this PS20 is popped from **current-path**, which remains empty; this signifies the completion of a cycle of the monitor.

6. Related Work.

In designing and implementing POPLAR a number of conceptual and technical decisions and choices had to be made. The following is an incomplete, though representative list.

1) how does one approach, and justify, construction of a multi-faceted system when little is known about the peculiarities of its components? Where is the starting point?

2) how might the problem of personality influences upon cognition be addressed?

3) within cognitive component, how are goals and plans related? How are they each related to such concepts as needs, drives, performance, etc.?

4) What is the structure of the planning module in cognitive systems? How is the scheduling of the cognitive system's activities performed?

5) What is the relationship between the use of internalized (canned) and newly created plans?

6) What is the relation between plan production and plan understanding?

The realization of the above and some additional problems was instrumental in the design stage. While not all of the decisions have been already made at this stage, our desire was to avoid design choices that would preclude or hamper a future improvement or extension.

None of the theoretical or design decisions were made without the influence of other, previous, related work. In this section we briefly review the bases for the various decisions as well as mention other work on the problems we faced.

Fundamental to the development of POPLAR was the approach to the task, faced by most cognitive modelers, of building a structure consisting of a number of distinct constituents, the details of many of which wer (and at present remain) unknown. How does one construct a global model when many of its components are uncertain, and each one is itself a mystery? Here we adopted teh attitudes advocated by Haugeland (1981), who suggests that it is appropriate to study an entire information processing system (IPS), consisting of several modules each of which (plus the IPS itself) is a black box, without first completing the study of the components; thus, we studied the cognitive actor even though we had not (and, obviously, could not) first provided an account for perception and performance.

Norman (1981) was very instrumental in specifying the tasks to be tackled in cognitive modeling. We also owe much to Anderson's (e.g. 1983) work on the architecture of cognitive entities. Sloman & Croucher (1981) discuss the introduction of motives, moods, attitudes and emotions in natural and artificial intelligent systems. Although no formalism is suggested for encoding this type of information, the general thrust of the approach is valuable for those who consider the introduction of certain personality characteristics into a class of AI systems. Wallace (1981) addresses similar problems in the context of learning.

Uhr & Kochen (1969) is an early work that addressed similar issues. Many of the important points for POPLAR have been anticipated in that work. Unfortunately, Uhr & Kochen's approach cannot be even called knowledge-based. It was an attempt to perform an important

piece of research with inadequate means.

Wood (1983) discusses planning in a dynamically changing world with multiple actors. Her system, AUTODRIVE, uses the world of the automobile driver as the domain. Although the design of the system depends too strongly on the implementation world, the idea of interaction between the actor and the world (in fact, the mere separation of the objective world and that of the actor -- through a program called SIMULATOR) is very fruitful.

Schank & Abelson (1977) and Schank & Lehnert (1979) informally discuss and catalog human (including interpersonal) goals. Carbonell (e.g. 1979) discusses the use of the concept of personal goals in the context of understanding stories. Wilensky (1983) also discusses everyday goals and metagoals, as well as various cooperative and competitive relations among them.

The relation between goals and plans is an interesting question that had to be addressed in our work. Our solution was to use this term only for top-level goals recognized by the goal-generators but made manifest in the system through the instantiation of a top-level plan. We did not use the concept of goals at lower levels in planning (i.e. we did not use the term 'subgoaling', cf. Lesk, 1984).

It is argued (cf. e.g. Barber (1983) or Berlin (1984)) that subgoaling is preferable to the use of 'canned' plans because if the latter are used then there is no possibility of ever achieving a goal in a non-standard way. But in the subgoaling approach, within the current state of the art, no unexpected results can be obtained either. To introduce these, one has to build a learning system, one capable of creating and not only recreating. But at present the planning of the subgoaling type remains no less 'canned' than the the 'forward' planning.

It seems that these two approaches to planning relate essentially in the same manner in which backward chaining relates to forward chaining in inference making. Our opinion is that the choice between the two is not strategically important and should reflect the peculiarities of the domain and other 'weak' considerations, so typical for AI.

Another important issue related to goals and plans is whether to build systems that in scheduling an action take into consideration the knowledge of how many different plans and/or goals will be furthered by it. The main empirical body of Wilensky's book (1983) is devoted to such issues. Cf. also Hammond (1983) for a philosophically related approach. Hayes-Roth & Hayes-Roth (1979) also want their planner to have this capability. Our position on this topic (cf. also Carver et al., 1984) is that in the type of planners we are building the goal cooperation or conflict does not play a role. We argue that to treat this topic as central in modeling planning in intelligent actors is similar to consider such non-everyday tasks as playing chess and solving differential equations central topics for AI. The latter methodological fallacy has been amply criticized.

General works on planning that immediately influenced this project include Stefik's work (e.g. 1981) on metaplanning and planner architecture. Hayes-Roth & Hayes-Roth (1979) describe a very rich planning domain and offer a good discussion of what the editors of **The Handbook of AI** (Cohen & Feigenbaum, 1982, p. 519; cf. also pp. 22 - 27) call opportunistic planning. It does not seem, however, that a non-trivial, involved implementation of the itinerary planner they suggest is possible.

Hayes-Roth (1984) is a definitive proposal concerning the architecture for planners. It addresses the control problem in AI systems as a whole. It also contains a comparison with other current proposals concerning control. In its architectural part this proposal (in fact, not only this proposal') draws heavily on the earlier work in the HEARSAY-II speech understanding system that introduced and popularized the blackboard architecture (cf. Erman et al., 1980).

The crucial idea of metalevel reasoning is discussed, with different emphases, in Stefik (1981), Hayes-Roth (1984), Wilensky (1983) and Genesereth (1983).

The basic architecture of POPLAR has a number of common points with that of Wilensky's planner (cf. Wilensky, 1983, pp.22-23). The two models, however, display major differences, notably in the attention paid in POPLAR to the problem of scheduling or in importance attributed to the idea of the independent representation of the objective world. Insufficient attention to scheduling and to describing the planning process at the system level were prominent among the criticisms in some reviews of Wilensky's book (cf. Russell, 1984; Berlin, 1984).

Many interesting ideas about scheduling can be found in Sathi et al. (1984) and Fox (1983).

Work on understanding plans in the POISE (Croft et al., 1983) and Argot (e.g. Litman & Allen, 1984) projects has helped in formulating some parts of our approach.

7. Status and Future Development.

POPLAR is a working system that generates and executes a relatively small number of plans in a rich, though simulated, environment. Its scheduling capabilities actually seem to transcend the immediate necessities of the domain. The system is designed in such a way that both domain planning and metaplanning are performed by one executor (that is, POPLAR can reason about its own actions). A number of features have been included that make POPLAR a model of a human planner in a real world.

At the same time, the possibilities of development and improvement that this basic system offers are probably even more exciting than experimentation with the current version of POPLAR. There are many points at which the system can be improved. Some of them are discussed below.

First (and simplest) of all, the POPLAR actor's knowledge about the objects, goals and processes both in the objective world and its own 'mind' can and will be augmented. In parallel, the control knowledge (rating and control functions) will be constantly adjusted and tuned, both through the introduction of additional character trait, mental state and situation parameters and through devising more appropriate ways of amalgamating them in the decision functions. Extensive experimentation with POPLAR will help to verify such decisions.

In parallel and in conjunction with the POPLAR project, these authors have been involved in designing a general model of human cognitive activity. Initial results of that research are reported elsewhere (Nirenburg & Reynolds, 1983; Reynolds & Nirenburg, in preparation). An aspect of that project extremely helpful to POPLAR is research aimed at deriving a set of 'primitive' character traits, motivations and mental states, such that weighted combinations of them will correspond to the 'higher-level' parameters (e.g. 'aggressiveness') that we would like to use in POPLAR's decision functions.

One can see that the above are actually two separate problems: 1) to extract primitives; 2) to express complex entities in terms of the primitives. It was decided to adapt the primitives suggested by Cattell (cf. e.g. Cattell & Child, 1975). Extensive psychological experimentation with humans is pursued in order to find answers to the second problem. The benefits of having a system that boasts psychologically valid (and not 'folk psychology'-based) control parameters are enormous and self-evident. And, therefore, this is one of the most immediate improvements we plan to make.

We also plan to add plan understanding to plan production. The world is inhabited by more than one cognitive actor (consider, for instance, the trolls in the current POPLAR). In order to behave correctly an actor must be able to discern plans of others. We believe that POPLAR's machinery will be able to handle plan recognition without the necessity to introduce major changes. An actor will maintain as many blackboards as there are cognitive actors around. It will assume that all other actors operate in the same manner. It will have beliefs about their character traits, etc. and will 'project' plans for them much in the same manner as it plans.

A logical extension to adding plan recognition is to introduce verbal behavior into POPLAR. There exist a number of interesting approaches to discourse analysis and plan understanding in dialogs (e.g. Allen, 1983b; Litman & Allen, 1984; Carberry, 1983; Reichman, 1984; etc.). A study in modifying POPLAR to involve verbal behavior and discourse analysis can be found in Nirenburg & Pustejovsky (1985).

The inclusion of multiple actors into the objective world can lead to the development of an experimental testbed for modeling conflict resolution, cooperation and many more important 'real-life' situations. The possibilities here are definitely substantial and quite unexplored.

The mechanism for modeling attention will undergo serious modifications, as will the treatment of time and the interaction between the actor(s) and the objective world.

And, finally, a most important avenue of improvement is the introduction of learning capabilities to the system. There are many modules in POPLAR where planning can be introduced; and there are many different types of learning to be studied. Some examples of this may be modifying the scheduling behavior depending on results of previous processing or after seeing somebody achieve a goal in a way not previously used; modifying beliefs about objects; being able to 'create' new plans, by analogy or otherwise; and many many more. This topic is one of the more complex ones, but any progress in this direction may have a very beneficial effect on the field of planning in AI.

Bibliography.

Allen, J., 1983a. Maintaining knowledge about temporal intervals. *Communications of ACM*, vol. 26, 832 - 843.

Allen, J., 1983b. Recognizing intentions from natural language utterances. In M. Brady & R.C. Berwick, eds., **Computational Models of Discourse.** Cambridge, MA: MIT Press.

Anderson, J.R., 1983. **The Architecture of Cognition.** Cambridge MA: Harvard University Press.

Barber, G., 1983. Supporting organizational problem solving with a work station. *ASM Transactions on Office Automation Systems*, Vol.1, No.1, January, 45 - 67.

Bates, P., J. Wileden and V. Lesser, 1981. A language to support debugging in distributed systems. University of Massachusetts COINS Technical Report 81-17.

Berlin, Daniel, 1984. Review of Wilensky (1983). *Artificial Intelligence*, vol. 23, 242-244.

Carberry, S., 1983. Tracking user goals in an information-seeking environment. Proceedings of AAAI-83. Washington, DC.

Carbonell, J., 1979. Computer models of human personality traits. Proceedings of IJCAI-79. 121 - 123.

Carver, Norman F., Victor R. Lesser and Daniel L. McCue, 1984. Focusing in plan recognition. Proceedings of AAAI-84. Austin, TX.

Cattell, R.B. and D. Child, 1975. **Motivation and Dynamic Structure.** NY: Wiley.

Cohen, P.R. and E.A. Feigenbaum (eds.), 1982. **The Handbook of Artificial Intelligence.** Volume III. Los Altos CA: Kaufmann.

Croft, W.B. and L. Lefkowitz, 1984. Task support in an office system. *ACM Transactions on Office Automation Systems*, Vol.2, No.3 197 - 212.

Croft, W.B., L. Lefkowitz, V. Lesser and K. Huff, 1983. POISE: An intelligent interface for profession-based systems. Conference on Artificial Intelligence. Oakland, Michigan, 1983.

Deering, M., J. Faletti and R. Wilensky, 1981. PEARL - a package for efficient access to representations in Lisp. Proceedings of 7th IJCAI, Vancouver, BC. 930 - 932.

Erman, L.D., F. Hayes-Roth, V.R. Lesser and D.R. Reddy, 1980. The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty. *Computing Surveys*, vol. 12, 213-253.

Fox, M., 1983. Constraint-directed search: a case study of job-shop scheduling. CMU Robotics Institute Technical Report 83-22.

Genesereth, M.R., 1983. An overview of meta-level architecture. Proceedings of AAAI-83. Washington, DC.

Hammond, K.J., 1983. Planning and goal interaction: the use of the past solutions in present situations. Proceedings of AAAI-83, Washington, DC.

Haugeland, J., 1981. The nature and plausibility of cognitivism. In: J. Haugeland (ed.), **The Mind Design.** Cambridge MA: MIT Press.

Hayes-Roth, B., 1984. A blackboard model of control. Stanford University Heuristic Programming Project Report HPP 83-38 (revised August 1984).

Hayes-Roth, B and F. Hayes-Roth, 1979. A cognitive model of planning. Cognitive Science, vol. 3, No.4.

Lesk, M., 1984. Universal Subgoaling. CMU PhD Thesis.

Litman, D. and J.Allen, 1984. A plan recognition model for clarification subdialogues. Proceedings of COLING-84. Stanford, 302 - 310.

McCue, Daniel and Victor Lesser, 1983. Focusing and Constraint Management in Intelligent Interface Design. University of Massachusetts COINS Technical Report 83-36

Nirenburg, S. and J. Pustejovsky, 1985. Plan Recognition and Production for Verbal (Discourse) and Non-Verbal Behavior. COINS Technical Report. University of Massachusetts

Nirenburg, S. and J.H. Reynolds, **1983. An architecture for a computer model of human cognitive systems.** Colgate University COSC **Technical** Report 4-83.

Norman, D., 1981. Twelve issues for cognitive science. *Cognitive Science*, vol. 4, No.1.

Reichman-Adar, R. Extended person-machine interface. *Artificial Intelligence*, vol.23, 157 - 218.

Reynolds, J.H. and S.Nirenburg, in preparation. The relationship between motivational traits, goal generation and plan selection

Russel, Daniel.M., 1984. Review of Wilensky (1983). *Artificial Intelligence*, vol. 23, 239-242.

Sathi, A., M. Fox, M. Greenberg and T. Morton, 1984. Callisto: an intelligent project management system. CMU CS working paper

Schank, R.C. and R. Abelson, 1977. **Scripts, Plans, Goals and Understanding**. Hillsdale, NJ: Erlbaum.

Schank, R.C. and Lehnert, W., 1979. The conceptual content of conversations. Proceedings of 6th IJCAI. 769 - 771.

Sloman, A. and M. Croucher, 1981. Why robots will have emotions. Proceedings of 7th IJCAI. Vancouver, BC, 197-202.

Stefik, M. 1981. Planning with constraints (MOLGEN: Part 1 and Part 2). *Artificial Intelligence* 16, 111-170.

Uhr, L. and M. Kochen, MIKROKOSMS and robots. Proceedings of IJCAI-69, Washington, D.C., 541 - 556.

Wallace, J.G., 1981. Motives and emotions in a general learning system. Proceedings of 7th IJCAI. Vancouver, BC, 84-86.

Wilensky, Robert, 1983. **Planning and Understanding**. Reading, MA: Addison-Wesley.

Wood, S., 1983. Dynamic world simulation for planning with multiple agents. Proceedings of IJCAI-83, Karlsruhe, Germany.

Appendix 1. Objects in POPLAR.

Representation of OBJECTS in POPLAR.

We present objects in two ways: first, in the way the object is stored in LTM, and second, from within a POPLAR run (as an annotated script). The difference is due to the inheritance of parents' properties by children in the hierarchy.

A.

```
(dbcr exp creature person       ; this is a PEARL header for a frame
        (id person)
        (type creature) ; CREATURE is the parent of PERSON
        (h-process-roles lisp ( (Take Who)
                                (Put Who)
                                (Find Who)))
                                ; the above are the roles in which an instance
                                ; of this type can appear in specified
                                ; processes by virture of its having properties
                                ; of a "human": humans can act as agents in
                                ; TAKE, PUT, and FIND
        (mental-state struct)   ; humans have mental states -- cf. the
                                ; default values in the script listing below
        (character-traits struct char-straits)     ; ditto
        (weapon-against ((sword 100 3) (knife 50 1) (rock 10 20)))
                ; POPLAR knows (believes) that weapons against people
                ; include swords, knives and rocks; the numbers (a b)
                ; indicate the efficiency of the weapon and the maximum
                ; range
        (power 50)      ; maximum
        (speed 50)      ; maximum
        (fearsomeness 25)       ; what is the level of fear that such objects
                                ; typically elicit in POPLAR (default: 25)
        (mass 55)
        (inventory lisp)        ; the objects this person is perceived by POPLAR
                                ; to be carrying
```

B.

```
POPLAR> person
        (person (id person)
                (type creature)
; ---------------------------------------------------------------------------
                (o-process-roles ((Find What)))
; this property is inherited by virtue of PERSON's being a
; descendant of OBJECTS: any object can occupy the "what" slot in Find,
; because finding mental objects is recollecting their representations in
; memory
; ---------------------------------------------------------------------------
                (shape nil)
                (color nil)
                (mass 55)
                (position nil)
                (p-process-roles ((Take What) (Put What)))
                (goal-parameters ((PS1 adv)))
; the above properties are inherited by virtue of a person being a descendant
; of PHYSICAL-OBJECTS; the goal-parameters slot specifies an instance of
; what goal is created when an object of this type is perceived. In this
; case the intuition behind the entry is that the appearance of a person
; spawns the creation of a goal instance of Preserve-Self-1, that is,
; persons are perceived by POPLAR as potential enemies.
; ---------------------------------------------------------------------------
                (edibility nil)
; this property is inherited by virtue of PERSON's being a descendant of
; +alive: nil is the default value with the semantics of "unknown"
; ---------------------------------------------------------------------------
                (c-process-roles
                        ((Eat Who)
                        (Ingest Who)
                        (Drink Who)
                        (Move Who)
                        (Attack (Who Whom))))
; the above properties are inherited by virtue of PERSON's being a
; descendant of CREATURE; creatures are considered by POPLAR to be able
; to be agents of eating, drinking, and moving, and agents and objects of
; attacking
; ---------------------------------------------------------------------------
                (weapon-against ((sword 100 3) (knife 50 1) (rock 10 20)))
                (power 50)
                (fearsomeness 25)
                (speed 50)
                (orientation nil)       ; this shows whether this particular person
                                        ; LOOKS at POPLAR at the moment of processing
```

```
; ------------------------------------------------------------ ----------
; the following are physical states (conceptually, they are part of
; the regulatory system)
                (hunger 0)
                (thirst 0)
                (fatigue 0)
                (injury 0)
                (h-process-roles ((Take Who) (Put Who) (Find Who)))
                (mental-state (nilstruct))
; character traits are a component of the regulatory system
                (character-traits
                        (char-traits    (greed 20)
                                        (pedantism 10)
                                        (hunger-tolerance 5)
                                        (thirst-tolerance 20)
                                        (fatigue-tolerance 20)
                                        (courage 25)
                                        (agression 40)
                                        (impulsiveness 30)
                                        (articulateness 40)
                                        (extravertedness 50)
                                        (locquaciousness 40)
                                        (curiosity 55)))
                        (inventory nil))
```

Appendix 2. Examples of PLAN representation in POPLAR.

```
(dbcr exp PLANS PS1
    (ID PS1)
    (Type PLANS)
    (Top-level-flag yes)
    (IS ((Plan-Selector Fight Wait-and-See))) ; Flee Hide
    (With (Actor Adversary))
    (COND ( (Plan-Selector '(Fight Wait-and-See) ; Hide Flee
                                    current plan)
                (Fight Actor Adversary)
                (Flee Actor Adversary)
                (Hide Actor Adversary)
                (Wait-and-See Actor Adversary)))
    (Preconditions (and (member 'Adversary (getpath ABB '(OBJECTS-PERCEIVED)))
                    ; Adversary is among the objects perceived by Actor
                        (or (= 'Actor 'self)
                            (and  (strcutrep Actor)
                                    (not (structurenamep 'Actor))
                                    (= (getpath (eval Actor) '(type) '(person))))))
                    ; Actor is either "self" or any instance of person
    (Rating-function (rating-fuc-PS1)))
```

```
(dbcr exp PLANS Plan-Selector
    (ID Plan-Selector)
    (Type PLANS)
    (Top-level-flag no)
    (IS (primitive))
    (Action-for-primitive (Schedule-of-plan 'list-of-plans 'calling-plan))
    (With (list-of-plans calling-plan))
    (Time 1)
)

(dbcr exp PLANS Fight
    (ID Fight)
    (Type PLANS)
    (Top-level-flag no)
    (IS (Find (Control1 (Find ! (Control2 (Get)))) (Control3 (Move ! Attack))))
    (COND ((Find Actor (getpath Adversary '(weapon-against))
                (getpath Actor '(inventory)))
           (Find Actor (getpath Adversary '(weapon-against))
                (getpath ABB 'OBJECTS-PERCEIVED)))
           (Get Actor (car result-find)
                (div (distance Adversary (car result-find))
                     (getpath Adversary '(speed))))
           (Move Actor (prog (weapon-range)
                     ; position to move to
                (cond ((<= (distance Actor Adversary)
                           (setq weapon-range
                                 (caddr (assoc (getpath (eval (car result-find))
                                                        '(type))
                                               (getpath (eval Adversary)
                                                        '(weapon-against))))))
                      ; if distance between Actor and Adversary is less
                      ; (or equal) than the range of the Actor's weapon
                      ; then Actor doesn't need to move toward Adversary
                          (return (getpath (eval Actor) '(position))))
                      (t (return (calculate-position Actor
                                                     Adversary weapon-range)))
                ))

           (Attack Actor Adversary (car result-find))))

    (Control ( (Control1 (Fight-Control1 Actor Adversary))
               (Control2 (Fight-Control2 Adversary))
               (Control3 (Fight-Control3))))
    (With (Actor Adversary))
    (Rating-function (rating-func-fight))
)
```

```
(defun Fight-Control1 (Actor Adversary)
        (cond ((not (= (car ABB.CURRENT-PATH.Status)
                       'succeeded))
               t)
    ; EITHER the last executed plan (which is Find) failed
               ((= (cadar Adversary.weapon-against)
                   (cadr (assoc (car result-find).type
                                Adversary.weapon-against)))
               nil)
    ; OR actor's current weapon is NOT the most efficient weapon
    ; against this adversary
               ((lessp
                   (div (times (distance Actor Adversary)
                               (diff  (cadar Adversary.weapon-against)
                                      (cadr (assoc (car result-find).type
                                                   Adversary.weapon-against))))
                        Actor.character-traits.impulsiveness)
                   fight-control1-threshold))
    ; OR even if the actor does not have the best weapon, he may decide not to
    ; look for a better one -- if the distance between him and the adversary
    ; is too small, if the actor is very impulsive, or if the weapon is not
    ; much worse than the best one
]

(defun Fight-Control2 (Adversary)
    (cond  ((null (cadr result-find)) t)
        ; no weapon was found in the actor's possession
           ((greaterp  (cadr (assoc (car result-find).type
                                    Adversary.weapon-against))
                       (cadr (assoc (cadr result-find).type
                                    Adversary.weapon-against))))
        ; the weapon found "around" is BETTER than
        ; the weapon in the actor's possession
]
```

Appendix 3. Examples of POPLAR rating functions.

A. The rating function for the Preserve-Self-1 goal (and top-level plan)

```
(defun rating-func-PS1 (actor adversary)

    (fix (div
                (times
                        (calculate-fear actor adversary)
                        actor.aggression)
                actor.courage)))

(defun calculate-fear (actor adversary)

    (fix (div
                (times adversary.orientation
                        (add adversary.mass adversary.speed)
                        adversary.power
                        adversary.aggr
                        adversary.fearsomeness)
                (times (fix (add1 (log (distance actor adversary))))
                        actor.courage
                        actor.power
                        (add actor.mass actor.speed)))))
```

B. The rating function for the Fight intermediate plan.

```
(defun rating-func-fight (actor adversary)

    (fix (div
                (times adversary.weapon-against.efficiency
                        actor.courage
                        actor.power
                        (add1 adversary.injury)
                        (expt actor.aggression 2))
                (times (calculate-fear actor adversary)
                        adversary.power
                        (add1 actor.injury)
                        adversary.fearsomeness
                        (add1 actor.fatigue)))))
```

Appendix 4. The History mechanism in POPLAR.

; this is the way HISTORY looks at the end of the example run of 5.4.

```
POPLAR> HISTORY
  ((Ingest0 Eat0 PS20)
   (Take0 Get0 Eat0 PS20)
   (Move0 Get0 Eat0 PS20)
   (Get0 Eat0 PS20)
   (Find1 Eat0 PS20)
   (Eat0 PS20)
   (Plan-Selector0 PS20))
```

Appendix 5. Blackboards in POPLAR.

Typical contents of the world and the actor blackboards.

```
POPLAR> WBB
        (World-Blackboard (ID WBB)
                          (NEW-INPUTS troll1 apple2 crocodile2)
                          (TIME (Base-Time (ID Time) (act-time 17))))

POPLAR> ABB
        (Actor-Blackboard (ID ABB)
                          (OBJECTS-PERCEIVED (troll2 sword1 gold-nugget2))
                          (STATES-PERCEIVED (hunger fatigue))
                          (AGENDA PS14 PS22 GTR4 Agenda-Scheduler)
                          CURRENT-PATH (find7 fight3 PS14)))
```

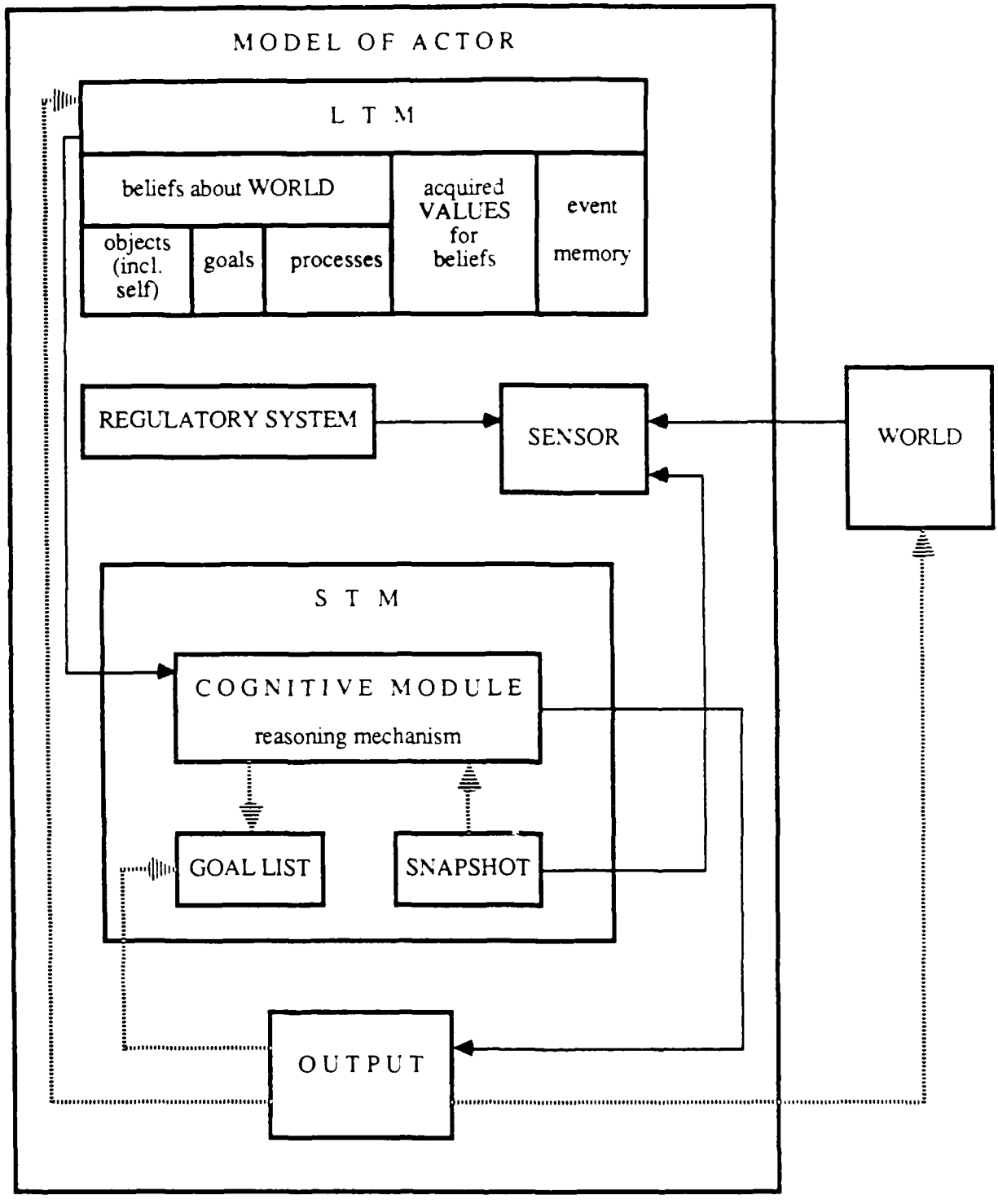


Figure 1. The conceptual architecture of POPLAR.

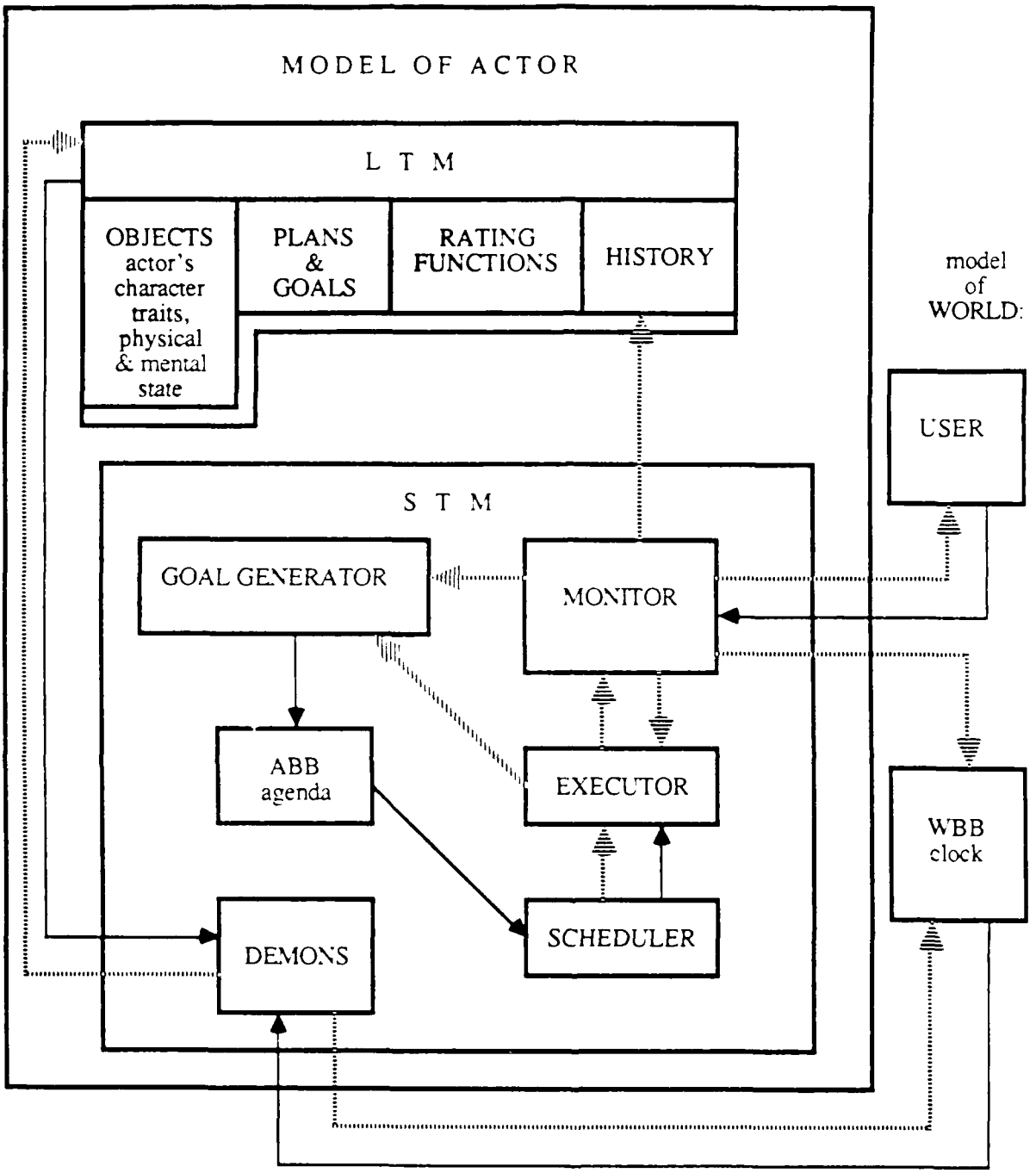


Figure 2. The system architecture of POPLAR 1.3

	TOP-LEVEL	INTERMEDIATE	PRIMITIVE
DOMAIN PLANS	PS1 PS2 GTR	FIGHT, EAT, GET, etc.	move, take, find, etc.
METAPLANS	GG as		as, gg-input, etc.

Figure 3. Classification of plans in POPLAR.

I ::= PS1 | PS2 | GTR | GG ('Goal-Generator') | as ('Agenda-Scheduler')
PS1 ::= FIGHT | HIDE | WS ('Wait-and-See')
PS2 ::= EAT | DRINK | SLEEP
GTR ::= {FIGHT | find} GET
GG ::= gg-input | gg-objects-perceived | gg-physical-states-perceived
FIGHT ::= find {find GET} move attack
HIDE :: find move
WS ::= do-nothing
EAT ::= find {find GET} ingest
DRINK ::= find {find GET} ingest
SLEEP ::= find do-nothing
GET ::= move take

Vertical bars separate disjointed elements; in practice, the 'or-ed' plans are chosen on the basis of their ratings through the application of a special metaplan we call the Plan-Selector, not shown in the grammar;

curly brackets enclose optional plans; the decision whether to execute the optional plan(s) is made on the basis of control functions that are stored in the parent plan and govern the processing of its IS slot;

plans shown in lower case are primitive.

FIGURE 4. A grammar of plans in POPLAR.

Plan Recognition, Knowledge Acquisition and Explanation in an Intelligent Interface

Year End Report
1984-1985

VICTOR LESSER, W. BRUCE CROFT, BEVERLY WOOLF

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

TABLE OF CONTENTS

LIST OF FIGURES 377

1.0 BASIC ISSUES 378
2.0 PLAN RECOGNITION/PLANNING 379
2.1 Integration of procedural and semantic databases 379
2.2 Focus of control mechanism 380
2.3 A system for knowledge acquisition 381
2.4 An interface monitor for a plan recognition system 382
3.0 INTERFACE TOOLS 382
3.1 Graphic display of task recognition 382
3.2 Natural language parsing 383
3.3 Natural language generation 385
3.4 Example generation facility 385
3.5 ThinkerToy: An Environment for Decision Support 386
3.6 Discourse analysis for a natural language interface 388
4.0 DISCOURSE PROCESSING IN A KNOWLEDGE ACQUISITION INTERFACE 390
4.1 The example knowledge acquisition system 391
4.2 The architecture of the system 393
4.3 Decision-making with discourse variables 396
4.4 Current status and conclusions 402
4.5 References 404

5.0 UNDERSTANDING DISCOURSE CONVENTIONS IN TUTORING 406
5.1 Abstract 406
5.2 Discourse conventions 407
5.3 Qualitative reasoning about discourse 410
5.4 Maxims of tutoring 413
5.5 Maxims and move-classes 417
5.5.1 Implications 417
5.5.2 Global implications 419
5.5.3 Managing discourse using global implications 422
5.6 Proposed Tutoring Discourse 425
5.7 Example Generation 428
5.8 Summary 429
5.9 References 430
5.10 Appendix 432
6.0 A KNOWLEDGE-BASED APPROACH TO DATA MANAGEMENT FOR INTELLIGENT USER INTERFACES 434
6.1 Introduction 435
6.2 Overview of POISE 437
6.3 The Knowledge Base 440
6.3.1 Task Descriptions 440
6.3.2 Object Descriptions 444
6.3.3 Tool Descriptions 446
6.4 Instantiation Management 448
6.5 A Frame-Based Implementation 452
6.6 Summary 454
6.7 References 456
7.0 POISE GRAPHICAL INTERFACE FOR TASK SPECIFICATION 458
7.1 Introduction and Objectives 458
7.2 Rationale for Implementation Decisions 459
7.2.1 Underlying Knowledge Representation 459
7.2.2 Graphical Interface Considerations 461
7.2.3 Machine and Language Used 462
7.3 Frame Tools, Utilities, and Demonstrations 462
7.3.1 Frame Tools 462
7.3.2 Utilities 463
7.3.3 Demonstrations 464
7.4 Future Work 465

LIST OF FIGURES

Figure 1: Example discourse to update an office database 392
Figure 2: The discourse comprehension and planning system 394
Figure 3: Portions of the discourse controller 395
Figure 4: Discourse variables (part 1) 397
Figure 5: Discourse variables (part 2) 399
Figure 6: Discourse variables (part 3) 400
Figure 7: Analyses of discourse conventions from the literature 411
Figure 8: Reading implications from utterances 414
Figure 9: Tutoring Maxims supported by move-classes 416
Figure 10: Implications bound to move-classes 418
Figure 11: Implicatures in text 419
Figure 12: Analysis in a tutoring interaction 420
Figure 13: Global implications about a student 421
Figure 14: Global implications about topic 422
Figure 15: Steps to manage discourse 424
Figure 16: A Student Program 425
Figure 17: Proposed tutoring discourse for the Program in Figure 9 427
Figure 18: The POISE Intelligent Interface 438
Figure 19: An Example Procedure Specification 441
Figure 20: Simplified example of instantiations during task execution 450
Figure 21: Frame Slot and EDL conventions 461

1. Basic Issues

The primary focus of the RADC interface effort is to support users in their interactions with an intelligent database and with each other, in the case of a distributed interface. The research effort has two complementary foci: 1) development of a planner, plan recognizer, and associated database that knows about the user's task and can begin to automate execution of his task; and 2) development of intelligent interface tools, such as natural language parsers and generators, and graphics facilities, to increase a user's ability to interact with the system. Both efforts are directed at enabling a machine to provide assistance to a user while describing its own actions and decisions.

The achievements for fiscal year 1984-5 are outlined below for plan recognition and interface work:

Accomplishments in Plan Recognition

Extended and implemented a plan recognition formalism to deal with a semantic database. The formalism can handle constraints derived from either procedures or objects;

Implemented a sophisticated focus-of-control mechanism that permits non-procedural specification of domain specific focusing heuristics and allows a more general recognition of a user's goal;

Developed a monitor for task specification that permits non-procedural specification of tools and makes the system architecture domain independent; and

Developed a graphic interface that allows the creation, modification and saving of procedural specifications to be done graphically.

Accomplishments in Intelligent Interfaces:

Implemented a natural language parser that interprets questions about tasks and procedures in the database;

Implemented an example-based on-line help system that provides customized examples in response to a user's request for help.

The research is described in detail in subsequent sections, divided into plan recognition and interface work. Reports and published papers are grouped together in sections four through seven.

2. Plan Recognition/Planning

POISE (Procedure Orientation Interface for Supportive Environments) was implemented for the domain of office automation between 1982 and 1984. Complete documentation of the goals of the system and detailed specifications of its architecture can be found in the POISE Project Report, 1984[1] has focused on integration of the semantic and procedural databases, construction of a monitor and graphic interface to the system, development of an knowledge acquisition system and design of a focus-of-control mechanism.

2.1 Integration of procedural and semantic databases

The object database has been implemented and the interface between it and the procedural database worked out. A complete description of this system is contained in Section 4.0, "A Knowledge-Based Approach to Data Management for Intelligent User Interfaces." The demonstration system is currently running with a

---

[1] University of Massachusetts/Amherst Technical Report No. 84-37.

simple database system pending the completion of the semantic database subsystem. The only outstanding problem is to develop a method of notifying the control mechanism (focuser) of interpretations which become inconsistent due to database constraint propagation. Such examples have not been properly handled by the existing demonstration system due to lack of an integrated semantic database.

The object-based concepts were implemented in SRL (Semantic Representation Language). We tested and rejected several frame-based languages before implementing the object database in a language called "Frame Kit," a personal version of SRL developed by Jaime Carbonell at Carnegie Mellon. Earlier systems were rejected because they were buggy, slow, or arrived without source code.

2.2 Focus of control mechanism

We have been working on a system to constrain large search spaces for plan recognition. The system[1] can rapidly and accurately recognize a plan based on the observation of a small number of steps. It uses hierarchies of plans to specify typical combinations of user actions and the goals they accomplish. By recognizing a user's actions in the context of this model of possible actions, POISE is able to provide intelligent assistance to a user, e.g., agenda management, error detection and correction, and plan recognition.

---

[1] Carver, N., Lesser, V., and McCue, D., "Focusing in Plan Recognition", *Proceedings of the American Association of Artificial Intelligence*, 1984.

A proposal to extend this focus-of-control mechanism into a new domain is close to being completed. We have been exploring the question of appropriate and available domains in which to develop the control mechanism, specifically the distributed database and vehicle monitoring task project at the University of Massachusetts as well as a software engineering environment.

2.3 A system for knowledge acquisition

We are developing a system for automating the process of knowledge acquisition. Specifically, the system is intended to receive portions of a dialogue between a domain expert and the knowledge engineer. It will decide how to change the database in accordance to the specifications of the domain expert. The motivation for this system and initial design are detailed in the POISE Project Review, 1984.

Progress in implementation of the knowledge acquisition system has included development and encoding of a "matching" metric to determine where new information best fits into the expert system's existing knowledge base. The system will take lexical input from a user and translate it into new or modified objects in a database, distinguishing whether the new object is 1) already in the database, 2) can be inserted pending modification to an existing object/procedure, or 3) requires definition of a new object/procedure in the database.

2.4 An interface monitor for a plan recognition system

A monitor for the POISE system has been built that recognizes a user's actions of specifying tasks to the system and reports these tasks to the planning and recognition systems. It also performs actions at the request of the POISE planner by mapping the action to a set of tool functions and then supervising the execution of those functions. The monitor is also able to recognize a definition of a new tool added to POISE. A description of the functionality of the system, its architecture and examples of monitor communication scenarios can be found in the POISE Project Review, 1984.

As a limited first step, we have implemented an interface to the formatted mail system. This system has been completed, though not yet generalized. It provides a general mechanism for integrating off-the-shelf tools into POISE.

However, the monitor is not yet included in the POISE demo system. This integration work will be taken up by a graduate student in the coming year.

3. Interface Tools

3.1 Graphic display of task recognition

We have developed a graphic interface to simplify a user's specification of procedures. See "POISE Graphical Interface for Task Specification" in Section 4.0 for a complete description. The interface is flexible in that it addresses general issues involved in the specification process, rather than being an interface for a particular domain of procedures. It is also well suited for depicting temporal ordering, since it

is graphic. With some additional work, it could be made easily learnable and implementable by naive users, and it could be made adaptable to different procedure domains.

We achieved two objectives in this area: 1) built a system that graphically represents tasks and 2) implemented an Event Description Language (EDL) generator. The former draws icons representing primitive subtasks and shows how they relate as complex subtasks involving shuffles or alternates. The Event Description Language (EDL) generator outputs the EDL formalism when given a name of a root frame of the procedure. The system includes a function that calculates the location of the innermost subtask (meaning most specific), given X & Y coordinates.

The system now allows cyclic creation of new procedures in which the user is queried for goal and subgoals until he says the task is complete. The "cycle" function will draw and query for modifications to the drawing until the modification is complete.

3.2 Natural language parsing

We use PLUM [Lehnert, 1983] to parse natural language input to the database. The parser now responds to 55 questions about tasks and actions related to the office environment, such as "Who purchased the VAX750?" and "From whom did we order the steel desks?" A description of the objectives, memory organization, and memory processes of this parser, as well as a listing of the 55 questions that the parser can handle, can be found in the POISE Project Review, 1984.

Our efforts with PLUM this year have included 1) providing additional flexibility for the parser as a front end to our database (it now parses only simple queries) and 2) providing PLUM with the ability to infer the intent of the user's requests based on a model of his activities, knowledge and the underlying semantics of the domain.

We have implemented a special parsing routine for noun phrase processing that interacts with memory. This work will be integrated into the general parsing effort to be used to generate responses to the user's questions about objects or procedures in the database. It is not as much an extension of PLUM as it is an optional facility for people working with the parser.

We have developed PLUM for the example generation system, EHELP, (Section 3.4) an on-line help system about commands in an operating system that customizes examples for the individual user and in some cases generates customized simulations of the effect of running the command. The parser for EHELP allows a user to describe a task in English or to use a name of an analogous command from a different operating system if he does not know the name of a command for which he is requesting help.

The long-term goal for the parser is to increase its effectiveness for naive users. PLUM supports natural language input for many systems in the COINS environment, such as legal reasoning and visions. In these cases, the parser provides a framework both for practical applications and for studying a variety of theoretical issues in linquistics.

3.3 Natural language generation

We are using MUMBLE [McDonald, IJCAI 85] to generate natural language explanations from the database. MUMBLE is designed to provide syntactically correct text from an expert system, independent of its internal knowledge representation. However, we have been unable to get MUMBLE to work in realtime for our expert system due primarily to the difficulty of porting systems between VAXes (where POISE resides) and Symbolics (where MUMBLE resides). MUMBLE is beginning to speak coherently about concepts in another expert system at the University of Massachusetts, a legal reasoner. In this system, both the text generator and the expert system reside on the same LISP-type system machine. We expect that when we move POISE to a LISP-type machine and develop a discourse-level translator between MUMBLE and our database, we will begin to achieve text generation.

3.4 Example generation facility

We have built a mechanism that provides explanations enriched with examples for an on-line help facility for VMS commands [Rissland et al, 1984]. The system determines a description of a relevant example, modifies the example to reflect the user's own files or previous questions, and then generates an example along with an explanation. This system accepts natural language input.

We are still adding new commands to EHELP and have made various other additions, including a mechanism for scrolling output on any terminal, and a mechanism to allow users within EHELP to mail comments about EHELP to the appropriate people.

Our goal is to 1) move example generation to the new intelligent interfaces database, 2) develop interactive tools by which a user can use examples to explore concepts, and 3) use more sophisticated modeling of the user (e.g., expertise level, past history with the system) to generate more appropriate examples.

3.5 ThinkerToy: An Environment for Decision Support

We are developing an environment for decision support that involves object/oriented modeling, incrementally extendable simulations, and integrated analysis tools. This environment is called ThinkerToy and is being developed on a Tektronix 4404 Pegasus with implemented Smalltalk. The interface will enable a user to perform effective reasoning through graphic tools. The user interface is a key aspect of the effort and is intimately linked to all aspects of the system. In particular, manipulation of graphical objects in the interface corresponds directly to actions for controlling the simulation, extending the simulation, carrying out analysis, and modifying the underlying code system.

The Smalltalk Machine is being used to build the basic structures for a class called DisplayEntity that will handle the basic metaphorical operations of:

1. object selection
2. object tracking (of mouse or other entity)
3. object tracing (leaving trail on screen)

4. object permutation (by binding to a point and re-constraining)

Several problems have been faced. The primary one is that a fast interface need demands that the system be carefully tuned and yet functionality of the interface requires that the same system be built generally enough to allow interplay between DisplayEntity and all parts of the system. The command DisplayEntity forms the lynchpin of a class of command, such as DisplayPanel, ControlPanel, and ProtocolPanel.

We are working on a good prototype of DisplayEntity based on the Model-View-Controller mechanism. The advantages of this mechanism are:

. Greater factorization of knowledge about the visual mechanics of each display item in Control and Display Panels. Control and View code is handled by each subview and the Model-View-Controller (MVC) handles the scheduling.

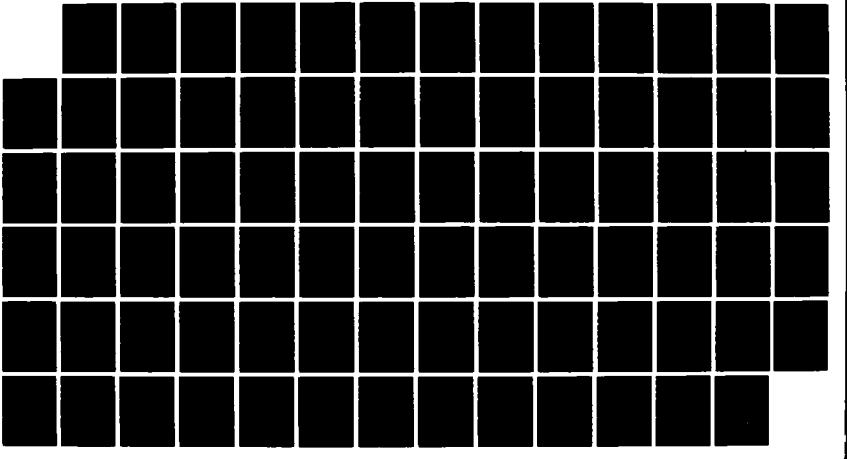
. Better access to sub-components of display. Display services for detecting boundaries, connecting graphs, form editing within displays, and compiling within displays.

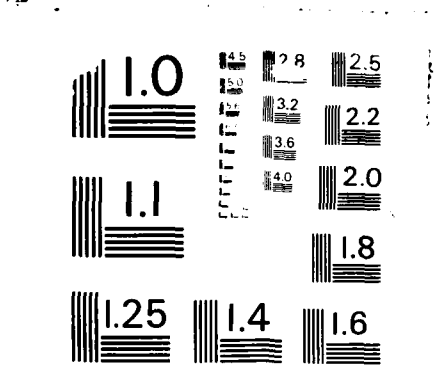
. Better flexibility in changing one's visual perspective to graphical data (a major goal of ThinkerToy). Thus the model ARRAY can either have a view of BAR to show as a bar chart or a view of LINE to show as a connected line chart.

The drawback is that there is now more to implement for each structure added to ThinkerToy (chart, text, number, scale, bar, etc.).

Our goal is to try out the basic function in building the objects of the system Text, Number, Array, Chart. We will soon begin to pull down DisplayPanels and ControlPanels for each item.

AD-A189 773 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM (NAIC) 5/5
REVIEW OF TECHNICAL T..(U) NORTHEAST ARTIFICIAL
INTELLIGENCE CONSORTIUM SYRACUSE NY J F ALLEN ET AL.
UNCLASSIFIED JUL 87 RADC-TR-86-218-V2-PT1 F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963

3.6 Discourse analysis for a natural language interface

We have developed a discourse manager for a large natural language system. The manager shares responsibility with the user for maintaining coherent and continuous discourse in the interface. It manages the flow of conversation, allows brief wanderings from a topic, and negotiates logical transitions from one topic to another. Discourse management consists of monitoring topics switches and co-ordinating communication between components of the natural language interface.

We have applied this technology in two ways as described by two papers in Section 4.0. The first paper, "Discourse Processing in a Knowledge Acquisition System" details a discourse parser for an on-line natural language system to assist a user in modifying a database and completes the document presented in the POISE Project Review, 1984. The second paper, "Understanding Discourse Conventions in Tutoring" describes our attempt to codify expectations and inferences made by people in a tutoring dialogue.

Our goal is to make discourse choices that allow the machine to anticipate a user's responses based on a model of the user, the complexity of the domain, and the current discourse history. We have begun to identify classes of discourse interactions and to implement a hierarchy of utterance types that are used to guide the direction of the discourse. We are also looking at the limits and task dependency of this mechanism.

Our work continues to establish a framework for machine response in the case of a variety of expert/novice interactions. We have also designed an intelligent authoring system that displays a variety of procedural, declarative, and heuristic knowledge to enable a teacher to modify and create topics in a tutoring system.

4. Discourse Processing in a Knowledge Acquisition Interface

Discourse Processing

in a

Knowledge Acquisition Interface

Beverly Woolf
James Pustejovsky
David D. McDonald
Susan Lander
[1]

Abstract

A natural language interface must track focus of attention and thematic distinctions across sentences in order to comprehend the user's input and to know how to respond appropriately. We are building a discourse system that recognizes shifts in discourse context and changes in the user's immediate focus. The system will respond in a manner and style that is consistent with the input utterance. This work is part of an on-going project to build natural language interfaces in a variety of domains.

---

[1] This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

We are building a knowledge acquisition interface to allow an untrained user to "talk" to a database in order to change it. The system works at two levels: at the level of an utterance, it comprehends the syntactic structure of the input and generates the surface structure for the output; at the level of the discourse, it comprehends the context and immediate focus of the evolving discourse and plans a consistent response. This paper focuses on the discourse component of the system and describes in detail how discourse processing is handled.

4.1 The example knowledge acquisition system

The target database for the knowledge acquisition discourse is a "bare-bones" representation of office procedures about travel forms, purchase requisition forms, people, and processes found in a university department environment [Croft and Lefkowitz, 1984]. The example in Figure 1 is a real-time on-line conversation between a secretary and the simulated[1] interface. The secretary is discussing changes to a procedure named *fill-out-authorization-form* in the original database. She is modifying properties such as when and where a travel authorization form should be submitted and the number of signatures required. Each piece of information is passed to a second system, KNAC [Lefkowitz, 1984], where the modifications are evaluated and corrections made to the existing procedure.

---

[1] The secretary's terminal was linked to another terminal and the system's utterances were "simulated" by one of the authors.

USER: *1. A travel authorization should be submitted to the accounting office*
2. 2 weeks prior to the date of departure.

SYSTEM: *3. Ok, so we submit the form two weeks early.*

USER: *4. Correct.*
5. The traveler must sign the form, along with the Principal Investigator.
6. Those are the only 2 signatures required, unless it is foreign travel.
7. If it is foreign travel, then the Dean and Provost's signatures are also required.

SYSTEM: *8. OK so we get 2 signatures if it is within America*
9. and two more signatures if it is foreign travel.

USER: *10. Right again.*
11. Do you have any questions about the travel authorization form?

SYSTEM: *12. What do I have to include on the form?*

USER: *13. The traveler s name, title, department.*
14. The date the form is typed.
15. The destination, date of departure, date of return, and means of transportation.

Figure 1: Example discourse to update an office database.

Several constraints have been employed by the system to comprehend this discourse. The first derives from the fact that a knowledge acquisition discourse is basically a teaching dialogue, i.e., the user is teaching the system about a new procedure or about modifications to an old one. As a result, the example contains several **"acknowledgement speech acts"** sprinkled throughout the dialogue (e.g., *OK* in lines 3 and 8, *Correct* in line 4, and *Right again* in line 10). These speech acts are recognized by the system as cue words that signify that successful "transmission" of new information has been made and, possibly, that a prior focus of attention has been terminated.

Another constraint comes from the **task-orientation** of the discourse and the fact that "subdialogues" reflect successive purposes or tasks [Grosz, 1980]. Portions of the discourse are concerned with a single modification and, in other portions, the system and user have cooperated toward accomplishing a specific task. For instance,

the utterances in lines 1-2 are recognized as related to submission of a form, lines 4-7 to signatures on the form, and lines 13-15 to the fields of the form. The last 3 lines are prompted by the system's goal to focus the user's attention on missing portions of the knowledge and to provide her with an opportunity to supply "filler" information as needed.

The next constraint conflicts with above task-orientation of the discourse and derives from the fact that humans work **opportunistically**, recalling details in a rather haphazard manner and pursuing each as it occurs. For this reason, we expect the order and occurrence of subdialogues to be largely **unpredictable**, to exhibit sudden shifts, and at times to require redefinition. For instance, line 12 is an example of how the system has led the user to focus on missing fields. Rather than allow a user to move the dialogue where she will, the interface will track topics and direct the user to refine or repeat information until all the slots are known.

4.2 The architecture of the system

In order to comprehend and generate this discourse, our system has been developed at two levels. At the discourse level the maxims of discourse [Grice, 1975] are handled by the discourse processor [Pustejovsky, in preparation] and the discourse controller [Woolf, 1984]. These two modules work together to comprehend some of the pragmatics of discourse and to understand the role of a single utterance in the *context* of the evolving discourse. At the utterance level the interpretation of input and the generation output are handled by the sentential parser [Lehnert, 1984] and the surface language generator [McDonald, 1983], respectively. These modules attend to details of sentential comprehension and to a final resolution of the syntactic

form of the output.

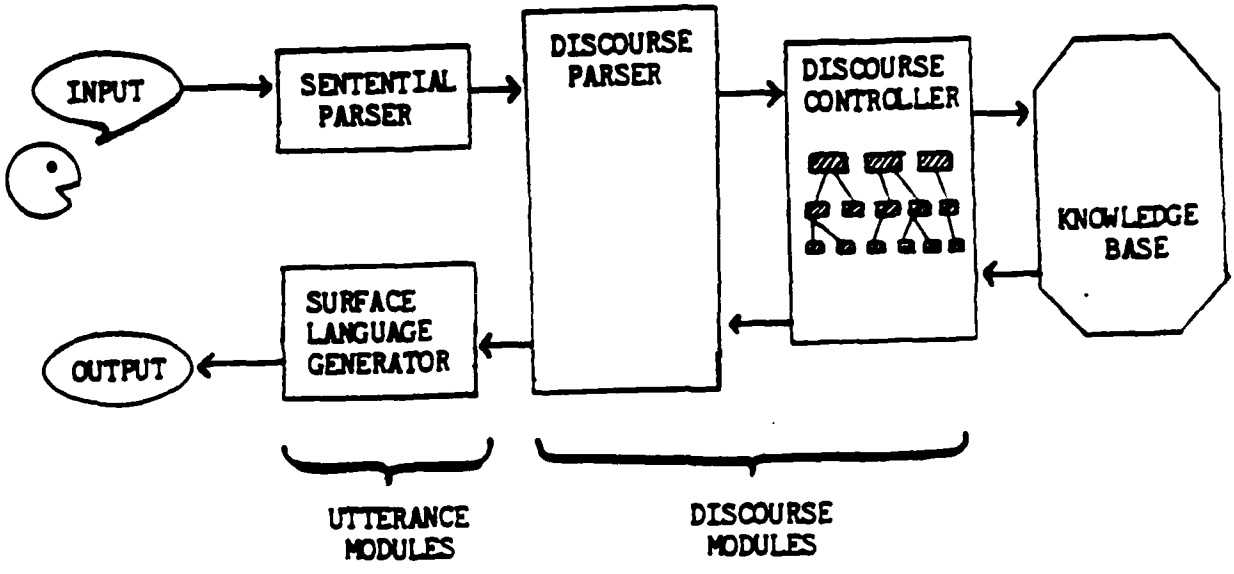


Figure 2: The discourse comprehension and planning system.

The discourse processor acts both as a deep level interpreter for incoming utterances and as a planner for outgoing responses. As an interpreter, it evaluates variables defined by the sentential parser and recognizes subtle conversational moves or inter-sentential shifts, such as change of topic or immediate focus. As a planner for the surface language generator, it adjusts the system's response to be consistent with the voice and mode of the evolving discourse. For example, if the discourse topic changes, the discourse processor might decide to change the mode or focus of the response, along with the topic. It might also decide to use ellipses, deletions or gaps in its response, or to respond in a curt or flippant way.

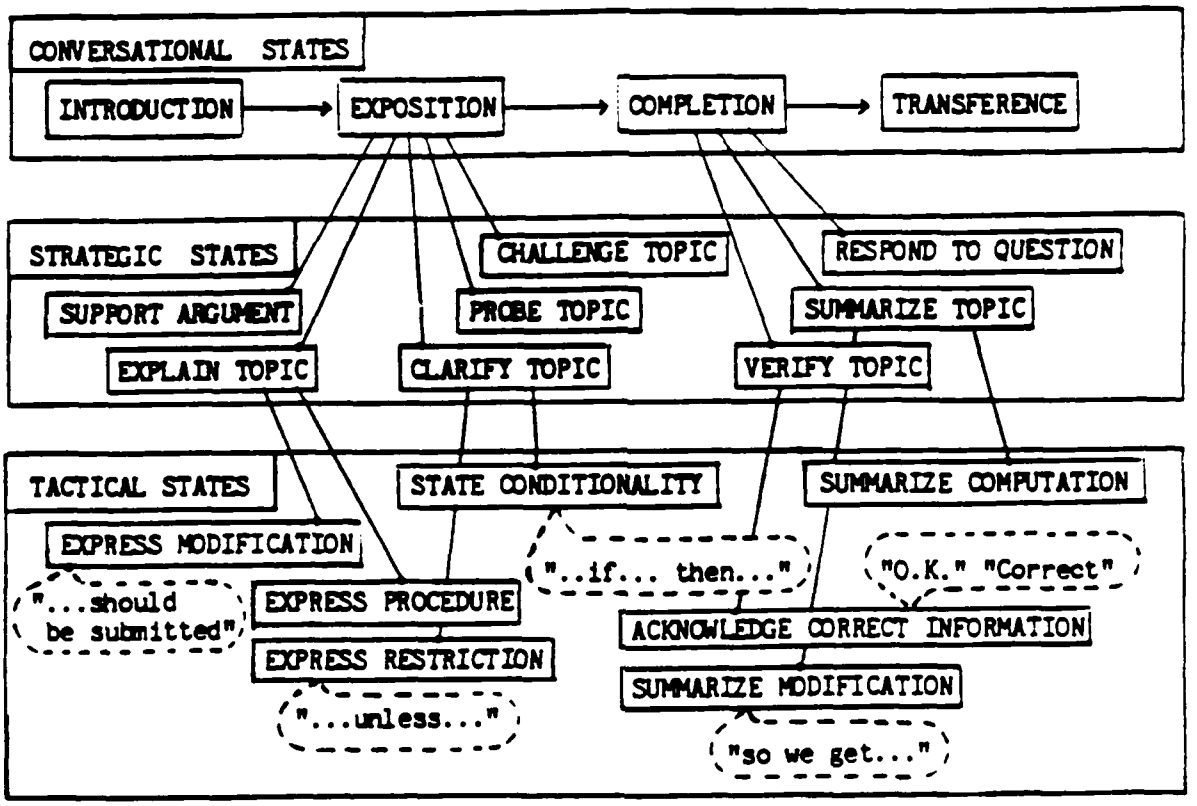


Figure 3: Portions of the discourse controller.

After interpreting the input utterance, the discourse processor passes a specification to the discourse controller, which plans the generalized response of the system. The discourse controller chooses a response which will be consistent with information accessed from the user, domain and discourse models. It reasons about whether it is opportune to follow a particular subdiscourse and whether to pursue a new topic [Woolf & McDonald, 1984].

The controller consists of a set of decision-units organized into three planning levels that successively refine the actions of the system response (Figure 3). Refinement at each level maintains the discourse approach dictated by the previous level and further elaborates the possibilities for the system's response. The controller

contains forty states, each organized as a LISP structure with slots[2] for functions that are run when the state is evaluated. The controller is structured like an augmented transition network (ATN) except that its paths are not fixed. Each default path can be preempted at any time by a "meta-rule" that moves the system onto a new path, which is ostensibly more in keeping with user or discourse history [Woolf, 1984]. The action of the meta-rule corresponds functionally to the high-level transitions observed in human discourse and the ubiquity of the meta-rules – the fact that virtually any transition between discourse states may potentially be preempted – represents an important deviation from the standard control mechanism of an ATN.

4.3 Decision-making with discourse variables

In this section, we provide an explicit view of the way decision-units and global variables interact during discourse processing and describe the interpretation and generation of a portion of the above discourse reproduced in Figures 4, 5, and 6. The example begins with the user's presentation of information, which is parsed into a conventional syntactic and semantic form. (Note: the syntactic structure of the parse has been omitted in the figures.)

The difficulty is that the sentential-based parser cannot know about thematic (i.e., old) and rhematic (i.e., new) distinctions in discourse. For instance, the definite reference *the date of departure*, in line 2, is recognized as referring to a specific date, but the sentential parse cannot bind it to a previously mentioned date. The

---

[2] The slots define such things as general specifications for text to be uttered, the next state, or how to update the user and discourse models.

USER: *1. A travel authorization should be submitted to the accounting office*
2. 2 weeks prior to the date of departure.

Unresolved entities from the input parse:

definite-reference: the date of departure

Interpretation of discourse context:

discourse_frame: **travel-event**
discourse_topic: travel_authorization_form
theme: travel_authorization
immediate_focus: submitted_to_the_accounting_office
rheme: λ x (x submitted-to A at T)
with A :== accounting_office
T :== 2_weeks_prior_to_T'
T' :== departure-date
mode: **EXPRESS-MODIFICATION**

Deep representation of response:

mode: **ACKNOWLEDGE-CORRECT-INFORMATION**
SUMMARIZE-MODIFICATION
pronominalization and anaphorization:
A_travel_authorization :== *the form*
submitted_2_weeks_prior_to
the_date_of_departure :== *submit two weeks early*

Surface text generation

SYSTEM: 3. OK, so we submit the form two weeks early.

Figure 4: Discourse variables (part 1).

discourse processor, on the other hand, establishes a discourse frame for travel-authorization-form that includes a representation for travel, agent and departure-date, similar to the "context space" of Reichman [1978]. The frame is used to make temporal inferences about the utterance and to resolve definite reference in terms of the larger **travel-event**. The processor makes an interpretation of discourse context for input lines 1-2 as indicated in Figure 4 under Interpretation of discourse context. It passes these variables on to the controller which determines the generalized form

of the response. The controller's default movement is from the state EXPRESS-MODIFICATION to the state ACKNOWLEDGE-CORRECT-INFORMATION and then to SUMMARIZE-MODIFICATION (See Figure 3).

The deep representation for the response in line 3 is summarized in Figure 4. The particulars of the response, i.e., how to summarize the information for generation, what to infer from discourse context or immediate focus, which anaphors or pronominalizations to use, and what topics to discuss given the user's beliefs, are determined by the discourse processor. The system's response in line 3 is equivalent to the input of lines 1-2, except that its intentionality and discourse context are different. For instance, the temporal adverbial "two weeks prior to the date of departure" is expressed as "two weeks early" in the response, because the processor *recognizes that a temporal pronominal* may be used, since this adverbial is part of the immediate *thematic* (i.e., mentioned) information. This decision is passed on to the surface language generator, along with a similar decision to omit the recipient of the form (i.e., the accounting office) in the response, since it was not part of the focused rhematic information (see also [McKeown, 1983]).

The surface language generator uses these decisions to realize the temporal frame as "two weeks early" and to omit the object frame in generating its expression "submit the form." The response is generated in the active voice although the input was in the passive voice, because the controller's default response voice for a knowledge acquisition dialogue is first person plural, a voice that is commonly used by a student in a teaching dialogue to indicate that she understands the new information.

USER: *4. Correct.*
5. The traveler must sign the form, along with the Principal Investigator.

Unresolved entities from the input parse:

definite-reference: *The traveler*
the form
the Principal Investigator

Interpretation of discourse context:

discourse_frame:	**signing-event**
theme:	the_form
immediate_focus:	the_traveler
rheme:	λ x (x sign THEME) (traveler and P.I.)
mode:	**ACKNOWLEDGE-CORRECT-INFORMATION**
	EXPRESS-MODIFICATION

USER: *6. Those are the only 2 signatures required, unless it is foreign travel.*

Unresolved entities from the input parse:

pronominal-reference: *Those*
it

Interpretation of discourse context:

theme:	the_2_signatures
immediate_focus:	2_signatures
rheme:	λ x (x is required)
mode:	**EXPRESS-MODIFICATION**
	EXPRESS-RESTRICTION

pronominal-resolution: *Those* :== **signing-event**
it :== **travel-event**

Figure 5: Discourse variables (part 2).

Analysis of lines 5-7 in Figure 5 shows the role of the discourse processor as a "deep interpreter." For instance, a word such as "form" of line 5 cannot be recognized by a sentential parser as referring to the "travel authorization" in line 1, unless an interpretation of "form" in line 1 had been stored as a part of the discourse interpretation of line 1 [Sidner, 1983]. Similarly, the word "the" in front of "form," "traveler" and "Principal Investigator" are recognized by the sentential parser

USER: 7. If it is foreign travel, then the Dean and Provost's signatures are also required.

Unresolved entities from the input parse:

pronominal-reference: *it*
anaphoric-reference: *also*
definite-reference: *the Dean*
the Provost

Interpretation of discourse context:

mode: **EXPRESS-CONDITIONALITY**

Head antecedent (*IF*)

theme: travel
immediate_focus: foreign_travel
rheme: foreign_travel

Head consequent (*THEN*)

theme: signature
immediate_focus: D_and_P's_signature
rheme: $\sim \lambda$ x (x is required)
pronominal-resolution:*it* :== **travel-event**
predicate-binding: *signature* :== **signing-event**
anaphoric-resolution:*also* :== *is also required*

Deep representation of response:

mode: **ACKNOWLEDGE-CORRECT-INFORMATION**
SUMMARIZE-MODIFICATION
RESTATE-CONDITIONALITY
pronominalization and anaphorization:
traveler_and_P_I_:== 2 signatures
D_and_P's_signature :== 2 more

Surface text generation:

SYSTEM: 8. OK so we get 2 signatures if it is within America and two more
9. If it is foreign travel.

USER: 10. Right again.

Interpretation of discourse context:

mode: **ACKNOWLEDGE-CORRECT-INFORMATION**

Figure 6: Discourse variables (part 3).

as usually referring to previously mentioned nouns, but since the referent entity is not clearly represented, except indirectly in the case of the first, no binding is made to a previous noun.[3] The discourse processor, on the other hand, has recorded that the topic of the conversation has not changed from the previous input and thus "form" is interpreted as the same "form" as that stored in the discourse interpretation for lines 1-2. In the cases of "the traveler" and "Principal Investigator" the discourse processor held no prior mention of the entities, yet because the reference is definite, the processor assumes that unique entities are referred to and accepts the requirement of two signatures without any verification. (The processor is not concerned with information verification.)

Interpretation of line 5 changes the discourse frame from time of submission, in line 1, to the signing of the form, and further changes the immediate focus to the number of signers of the form. In line 6, reference to "those" as a demonstrative Noun Phrase (NP) confuses the sentential parser by a suggestion that it refers to a previously mentioned noun, which is not the case. "Signatures" as an NP has not yet been mentioned and reference to it is an example of a "deep anaphoric" reference [Hankamer & Sag (1976, 1984)], where the antecedent is not present in the same syntactic form. The obvious interpretation of this NP is as a pronominalization of a "result nominalization" (i.e. the result of signing), anaphoric to the event of signing, mentioned in line 5. The processor is able to recognize the relationship between the **signing-event** and **signature** based on a comparison of slots in the two

---

[3] Definite description containing restrictive clauses, however, are interpreted by a sentential parser and given a reference, since the relative acts as a predication identifying the head.

frames.

A second anaphoric reference is resolved in line 6. The "it" before "foreign travel" refers to the entire **travel-event** frame introduced in the first line of the discourse. Because travel authorization is still the major discourse topic and because the predication of the pronoun makes explicit reference to travel (*i.e. foreign travel*), the processor had no difficulty selecting the correct referent.

A form of deep analysis of an antecedent group is required to interpret the word "also" in line 7, Figure 6. This word functions in this context as a "predicate anaphor," making reference to a previously mentioned event predicate – namely, "to be required." Interestingly though, the actual antecedent was not so worded in line 5, but was realized as "x must sign y." This deep anaphoric connection was made by the processor through a similar procedure to that mentioned above for NPs.

In line 8, the controller decides to acknowledge the user's correct utterance and again the discourse processor is succinct in its response and summarizes on the basis of the immediate focus of the previous utterance: number and type of modified signatures to be collected.

4.4 Current status and conclusions

The separate modules described in this paper are fully implemented, with the exception of the discourse processor, and the individual modules are being linked together. The system can parse the user's input utterances and is beginning to reason about the responses.

Though we have isolated some constraints to comprehend and plan discourse, our knowledge of conceptual analysis and planning at the discourse level is presently rather primitive. For instance, we do not understand how to represent a topic through many views and modifications of focus, nor how to anticipate the reoccurrence of an abandoned topic. Nevertheless, our handling of thematic and rhematic distinctions over clusters larger than a sentence allows us to perform some recognition and tracking operations not possible before. For example, we can resolve pronominalizations and track focus of attention across sentences.

4.5 References

Croft, B. & Lefkowitz, L., "Task Support in an Office System," *ACM Transactions on Office Information Systems*, Vol 2, No 3, July 1984. Also available as Computer and Information Sciences Technical Report #84-24, University of Massachusetts, Amherst, Mass., 1984.

Grosz, B. J., "Focusing and Description in Natural Language Dialogues," in A. Joshi, et al. (Eds.), *Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting*, Cambridge University Press, Cambridge, 1980.

Hankamer, J. & Sag, I., "Deep and Surface Anaphora," *Linguistic Inquiry*, Vol 7, No 3, 1976.

Lefkowitz, L., "Knowledge Acquisition for Expert Systems," unpublished Ph.D. proposal, Department of Computer and Information Sciences, University of Massachusetts, Amherst, Mass., 1984.

McDonald, D. D., "Natural Language Generation as a Computational Problem: An Introduction," in M. Brady & R. Berwick (Eds.), *Computational Models of Discourse*, MIT Press, Cambridge, Mass, 1983.

McKeown, K. R., "Focus Constraint on Language Generation," International Joint Conference on Artificial Intelligence, 1983.

Pustejovsky, J. D., "Planning, Discourse Interpretation and Plan Recognition," in preparation.

Reichman, R., "Conversational Coherency," in *Cognitive Science*, Vol 2, No. 4, 1978.

Sag, I. & Hankamer, J., "Toward a Theory of Anaphoric Processing," in *Linguistics*

and Philosophy, Vol 7, No 3, 1984.

Snider, C. L., "Focusing in the Comprehension of Definite Anaphora," in M. Brady & R. Berwick (Eds.), *Computational Models of Discourse*, MIT Press, Cambridge, Mass, 1983.

Woolf, B., *Context-Dependent Planning in a Machine Tutor*, Ph.D. Dissertation, Computer and Information Sciences, University of Massachusetts, Amherst, MA, 1984.

Woolf, B. & McDonald, D. D., "Context-dependent Transitions in Tutoring Discourse," *Proceedings of the National Conference on Artificial Intelligence*, 1984.

5. Understanding Discourse Conventions in Tutoring

Understanding Discourse Conventions

in

Tutoring

Beverly P. Woolf
David D. McDonald

COINS Technical Report #85-22

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Published in the *Proceedings of the Expert Systems in Government Symposium*, sponsored by IEEE & Mitre Corp, Oct 23-25, 1985, McLean, Virginia.

5.1 Abstract

Speakers have expectations about listeners that enable them to produce coherent discourse. These expectations should be incorporated into machine tutors so that they too can generate expectations about their student users. We intend to show how expectations can be used to anticipate a user's *choice* of responses based upon the dynamics of the speaker/listener interaction. The paper describes a way to

[4] "This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

formalize the constraints and operations in discourse and how to use these constraints to transform interpretation and speech act knowledge into computational elements, such as plans and rules.

5.2 Discourse conventions

One of the largest theoretical stumbling blocks in the design of effective machine discourse systems is the lack of an adequate representation or understanding of discourse conventions. *Human* speakers employ subtle linguistic cues to shift topics or provide supplementary knowledge. Listeners use these cues to set up expectations about the underlying structure of the discourse and to relate current utterances to preceding ones. The listener's expectations are what the speaker tries to anticipate and to deliberately control.

The aim is to build a machine speaker that represents these conventions and responds to its user based on inferences about a model of the user or the discourse history. Early computer discourse systems controlled the flow of discourse producing canned texts that were typically the same regardless of the user's knowledge or the discourse history.[1,2] More recent interface systems have begun to tailor their responses to the user and discourse context.[3,4] The basic problem in designing machine discourse is how to make inferences about the user and how to have these inferences govern the form of the text produced*.[1]

---

[1] *We recognize that a machine cannot know with certainty what a listener knows, neither can a listener know what a machine knows; a machine cannot be omnicient or clairvoyant. However, the machine can deduce, on the basis of evidence, something about what the listener assumes. These assumptions can be used to govern the form of the text generated.

For instance, the adjustments that a *computer tutor* would make are dependent upon its specific experience with a student and a variety of experiences would lead to a variety of responses. Thus we would want a computer tutor to interact with a knowledgeable student in a way that is fundamentally different, both in style and content, from the way it would engage a confused one. It is not intended that the computer simply produce correct answers in response to a student's wrong answers; rather before responding to a wrong answer, the machine should resolve issues such as:

>> when and how to stop to explain the wrong answer;

>> whether it is preferable to explain the error or to start a lengthy exploration of the student's knowledge;

>> whether to allow uncertainty about the student's knowledge to persist temporarily while it explores a potential misconception; and

>> how hard it should work to understand why a student answered a question incorrectly or how much effort should be exerted to resolve questions about the student's presumed knowledge or misconceptions.

Though many areas of research on understanding discourse conventions are interesting and several problems are ripe for a solution, we have focused on the role of the speaker because we want to study discourse in the context of tutoring. In tutoring, perhaps more than in other types of discourse, the speaker (the tutor) chooses conversational moves based on the responses of the student. There are many occasions in which a tutor will interpret what a student says and "read into" the answer additional material to update his current model of the student's knowledge. Based on these considerations, a human tutor would adjust the discourse; a machine tutor should do the same. Tutoring provides us with a rich, well-contained field in which to study discourse conventions from a speaker's point of view.

A second reason to work in tutoring is because of the wealth of research on language and tutoring in our environment at UMass. One research effort has focused on natural language comprehension,[5,6] generation,[7] discourse control,[8] and legal reasoning.[9] Another effort has focused on tutoring discourse[10] and a related effort at Yale has focused on student errors[11] and the learning and teaching of Pascal looping constructs.[12,13] As a result of this extended research environment we have been able to formalize knowledge about human tutoring protocols, understand the epistemology of Pascal looping constructs and have a realistic way to accumulate a rich model of the user. Therefore, we have a domain where the system can select the appropriate content to discuss with the student based on an understanding of its audience, in addition to in-depth knowledge of language and tutoring.

This paper discusses several research areas being pursued, including problems to be solved, recent research in the area, and conclusions that might be drawn about discourse conventions as a result of our studies. It also presents an example of how this computational model is being used to build a robust tutor for Pascal programming. Some of the research areas to be discussed are:

- o discourse control - how to focus on appropriate topics, errors or examples;
- o knowledge representation - how to create a data structure for the codification and interpretation of utterances; and
- o natural language generation - how to produce appropriate text for the situation.

5.3 Qualitative reasoning about discourse

Discourse is often described in qualitative terms:

"the speaker was [helpful] [abrupt] [angry]"

"the student was [confused] [unprepared] [sharp]"

"the topic was [important] [understood] [trivial]"

Awareness of the *effect* of an utterance on the overall interpretation of the discourse is also described in qualitative terms such as:

"the example was useful"

"the argument was weak"

Figure 7 contains other analyses of discourses from a psychologist, computational linguist and psychiatrist. In each case, the researchers have teased out implicit rules of discourse based on how a speaker should interpret his listener's level of knowledge or understanding. The impact of these rules suggests that people would be better speakers or tutors if they followed implicit rules. To represent these rules in the machine tutor and to enable it to demonstrate the same aspects of good discourse conventions alluded to in the analyses, is the present goal of our research.

Toward this end, we have begun to capture several of the features we recognized in the analyses of Figure 7. For example, the analyses refer to inferences (in italics) about the student's prior knowledge, or the "mutual" knowledge of the two conversants. Our system will recognize qualitative inferences such as when a ***topic is generally known**, ***student has background information**, or ***student is confused**.

From Analysis and Synthesis of Tutoring Discourse:[14]

> [A tutor] builds on what *the student already knows* [and] can question him about *his previous knowledge*. Then he can teach new material by relating it to that *previous knowledge* [pg 50].
>
> [A tutor] can respond directly to student errors, . . . question him to *diagnose the confusion* and can provide *relevant information* to *straighten him out*. [pg 50].
>
> The question *raised the issue* of . . . moving [the discourse] to a *deeper level of analysis* than made so far [pg 67].

From Plain Speaking: A Theory and Grammar of Spontaneous Discourse:[15]

> Much of the implicit *knowledge speakers and listeners share* is knowledge of the particular components of various conversational moves – what kinds of utterances must be made in order to fulfill various discourse functions [chapter 3, pg 1].

From Parental Communication Deviance and Schizophrenia: A Cognitive-Developmental Analysis:[16]

> A failure on the part of the speaker to *establish and maintain a shared focus of attention* with one's listener [pg 68].
>
> A tendency to *equivocate concerning one's commitment* to one's statements and a tendency to *vacillate concerning the content* of one's statement [pg 68].
>
> A *lack of specificity* with regard to the referent, *unexplained contradictions* . . . inappropriate responses suggestive of a failure to grasp the *intent of a question by the interlocutor* [pg 62].
>
> [A failure] to take into account the *cognitive needs* of the listener [pg 62].

Figure 7: Analyses of discourse conventions from the literature.

Also represented in the analyses are qualitative inferences about knowledge, particularly mutual knowledge:

"what the student already knows,"

"deeper level of analysis,"

"shared focus of attention."

These inferences are not defined or explained in the analyses and their casual use suggests a degree of subjectivity about quantities such as "knowledge," "confusion," or "attention." In addition, to understand these metrics the reader is expected to understand processes such as "*building* on what a student knows," "*raising* issues," and "*establishing* a shared focus of attention."

Representing these complex discourse conventions and metrics requires using qualitative expressions of knowledge. There is evidence from other fields that qualitative reasoning and representations are useful: e.g., teaching,[17] Artificial Intelligence (AI),[18] and the domain of physics.[19] Tracing qualitative inferences in a discourse model will be relatively intractable, compared with, for example, tracing speech acts. Qualitative inferences will be multiplexed between and within other streams of inferences, some being initiated or continued while others are simultaneously being started. The result is that the intent of a particular stream of inferences can become confounded. Yet, we suggest that it is worth the effort to try to make qualitative inferences because they provide a more powerful representation of the intention of the speaker than do speech acts. In particular, they are more predictive of subsequent utterances and can be used to propose and elucidate a speaker's intent or the direction of the discourse. We suggest that tracing implications to evaluate the effect or goal of a discourse provides a sound framework for understanding discourse.

5.4 Maxims of tutoring

We suggest that tutoring consists of following certain maxims of discourse conventions (in the same sense used by Grice[20]) and we analyze research such as that in Figure 7 to identify these maxims. We expect to be able to evaluate the reasonableness of the tutoring discourse we produce by recognizing whether the maxims are satisfied. In this section we define some tutoring maxims and outline how we intend to monitor discourse based on a notion of maxim satisfaction.

In order to model the qualitative effect of utterances we first define conversational move-classes as groups of utterances that have the same rhetorical effect, such as **question topic**, **summarize topic**, **acknowledge correct answer** and **provide example**. We suggest that a tutor's choice of conversational move indicates his (its) "intention" in the sense that a move sets up expectations in a listener. For instance, a conversational move such as **make accusation** typically would elicit negative responses from a listener. For instance, consider some queries a tutor might pose to a student about loop execution in a Pascal program, as suggested in Figure 8. Each sentence has a similar locutionary force, yet each conveys a different intention on the part of the speaker. Further, there is a continuum such that a tutor may couch his statements at any place along the higher end. The implication drawn would be of close attention, even commitment, to the student. On the other hand, a statement at the lower end would imply non-commitment, non-involvement and possibly antagonism. Relative to the four utterances above, we say that use of a phrase representing a certain point on the scale implies that the tutor chose not to phrase the utterance by another expression lower on the list. This reasoning on the

(*provide-example*)
(*question-hypo*)
If the input is 10, how many times would your loop execute?

(*question-topic-value*)
Do you know how many times your loop would execute?

(*make-a-claim*)
I bet you don't know how many times your loop will iterate.

(*make-an-accusation*)
You couldn't possibly understand loop execution.

Figure 8: Reading implications from utterances.

---

part of the listener is licensed by the Gricean[20] maxim of manner. Grice has defined very general maxims for discourse, that are evocative, yet not detailed enough to provide a basis for a computational theory of discourse by themselves.

Our goal is to propose a computational model of tutoring discourse that elucidates and refines these maxims and links them with specific conversation moves. Ultimately inferences about conversational moves will be used to guide the system's choice of utterances. The tutoring maxims that we propose are derived from Gricean maxims for discourse and are tailored for tutoring. They include:

Quality: be committed and interested in the student's knowledge;

be supportive and co-operative;

do not take the role of "antagonist"

Quantity: be specific and perspicious;

use a minimum of attributes to describe a known concept;

Relation: be relevant;

find a student's threshold of knowledge;

bring up new topics and viewpoints as appropriate

Manner: be in control;

allow a student to determine a new topic;

allow context to determine a new topic.

Figure 9 further discriminates these maxims in terms of move-classes that support each one. Maxims are listed on the left and the sequence of move-classes that supports them on the right. By being attentive to moves during discourse, a system can monitor its own behavior and guide subsequent moves so as to be consistent with the maxims of good tutoring. The system can identify maxims on the left, and invoke the move-class on the right that are associated with them. For instance, if the system plans to be more organized, it can **outline topics, introduce topics, terminate topics**, and then **review topics**. Alternatively, if the system needs to record the "effect" of its actions on the listener, it can list the actions taken by the tutor and determine if its own actions are consistent with certain maxims. For instance, if the interaction with the student could be described as an ordered set of utterances, such as **question student, acknowledge answer, propose misconception**, and **provide example**, the overall effect of the actions could be to determine the student's threshold of knowledge. Whether or not that threshold was determined is a non-trivial, and as yet unanswerable, question.

The table in Figure 9 can be read in two directions: from left to right it allows the system to select a maxim and plan subsequent tutoring discourse by invoking the associated sequence of move-class; from right to left it provides an abstraction of the system's activities so that the effect of the system, in terms of the expectation of the listener and the maxims of good tutoring, can be expressed.

Maxims	Conversational move-classes
Be co-operative:	
–work with student	
	explain topics
	summarize topics
	clearly terminate topics
	review or repeat topics
	release control of dialogue
Be committed:	
–show interest	
	acknowledge answer
	explain topics
–support student	
	outline topics
	introduce topics
Be relevant:	
–find student's threshold	
	question student
	evaluate student hypotheses
	propose and verify misconceptions
–teach at threshold	
	provide analogy example
	summarize topic
Be organized:	
–structure domain	
	outline topics
	introduce topics
	terminate topics
	review topics
–complete information	
	clearly terminate topics
	teach subtopics after topic
	teach attributes after topic
	teach subgoals after goal
Be in control:	
–strictly guide discourse	
	introduce topic
	describe topic
	question student

Figure 9: Tutoring Maxims supported by move-classes.

5.5 Maxims and move-classes

In order to computationally associate maxims with sequences of move-class, we need to make inferences about the qualitative effect of each move-class on the discourse. To do this, we suggest the effect that each move-class has on discourse entities, such as topics or a student's knowledge. Each conversational move is defined as a data structure and two inferences are made from it. The first inference or *implication*[2] is linked *directly* to a move-class. It represents an assessment made about the move-class itself and is fixed and non-negotiable. The second kind of inference or *global implication* is linked to **indirectly** on sequences of move-classes. It represents an inference made about the effect of several move-classes and is volatile over the life of the dialogue. Global inferences are dynamically modified by the sequence of move-classes. Each inference type is discussed below.

5.5.1 Implications

Implications are bound to the move-class itself. They exist independent of the "truth" or "meaning" of the utterance and define what the listener receives in addition to the spoken words. In our model, a qualitative implication bound to the move-class is placed on a stack whenever its move-class is invoked. Figure 10 lists the implications bound to two move-classes, **question topic** and **present topic**. For instance, if a tutor questions a student about a topic, the implications of this are that the tutor 1) knows (or is trying to learn) the student's threshold of knowledge,

---

[2] *An implication was originally called an implicature by Grice and was attached to specific words, not to groups of words.

Typical objects in our ontology

(define-move-class QUESTION-TOPIC
Evidence:
Q+ *topic is important*
Q+ *topic is within threshold of knowledge*
Q+ *topic is learnable through discourse*)

(define-move-class PRESENT-TOPIC
Evidence:
Q+ *topic is generally known*
Q+ *topic is background information*
Q++ *topic is less important*
Q++ *topic is impact material*)

Figure 10: Implications bound to move-classes.

2) assumes the student can answer the question, 3) thinks the topic is important or is learnable through the discourse. These implications can be assumed by a listener independent of the content of the query.

We speak of implications in the same sense as Grice's implications, but in reference to sequences of words perceived as a single conversational move. Grice's implications originally referred to inferences made over single words. For example, the italicized words in Figure 11 have explicit implicatures. The word *and* in the first sentence carries an implication that the activity of going to jail preceded, and possibly caused the second activity, that George became a criminal. The use of the word *tried* in the second sentence carries with it an entailment that Millie failed to swim the English channel, and the use of the phrase *one leg* in the third sentence, implies that the speaker *does not* in fact have two legs.

1) George went to jail *and* became a criminal.

2) Millie *tried* to swim the English channel.

3) I have *one* leg.

Figure 11: Implicatures in text.

Implications, as we use them, define each participant's common-sense reasoning about a conversational move. They include the desiderata normally accepted by a rational discourser. We would like to think that implications embody a speaker's motivation, intention, and involvement in the discourse.

5.5.2 Global implications

Global implications are based on extended reasoning about sequences of move-classes. They include assessments such as ***student is confused**, ***topic is known** or ***misconception is resolved** and are modified with each new tutor/student interactions. Global implications are uncertain and represent the system's best estimate about the state of affairs of knowledge of the student or topic at the current time. Whereas, implications were known with certainty at the time a move-class was invoked, global implications require reasoning under uncertainty to deduce which one of a number of competing global implications might take effect. Reasoning with uncertainty must allow for the accumulation of support for or against a number of global implications.

Tutor: Do you know that *GRADE* in line 8 is a control variable?
Student: Yes

; *IMPLICATIONS*
; *student_has_definitional_knowledge
; *topic_is_generally_known
; *topic_is_learnable_elsewhere
; *topic_is_background_material

Tutor: Good. What is the value of grade before leaving the loop in line 13?
Student: 9999

; *IMPLICATIONS*
; *topic_is_generally_known
; *topic_is_learnable_elsewhere
; *topic_is_background_material

; *GLOBAL IMPLICATIONS*
; *student/domain_agreement
; *student_knows_the_topic

Tutor: That's right. What is the value of grade after leaving the WHILE loop, in line 13?
Student: I don't know.

; *IMPLICATIONS*
; *tell_tale_signs_lack_of_knowledge
; *student_does_not_know_the_topic

; *GLOBAL IMPLICATIONS*
; *topic_is_on_student's_threshold
; *student/domain_disagreement
; *student_is_confused

Figure 12: Analysis in a tutoring interaction.

Figure 12 presents an example of how global implications can be inferred over the course of a tutoring dialogue. In the example, the tutor's goal is to determine the breadth of the student's understanding of primitive topics about Pascal loops. Three questions are presented that might be asked of a student who had submitted an incorrect Pascal program. After the first correct response certain immediate inferences can be made; the student has definitional knowledge of the topic, the

topic is generally learnable through other efforts (i.e., textbooks or lectures), and the topic was studied as background material. After two correct answers, the tutor has reinforced its initial evaluation of the student's knowledge but now is licenced to make more extensive inferences about the student or the topic. In this case, the global implication might be that there is some agreement between the student's information and the domain knowledge base. This inference is possible because evidence from the additional correct answer provides support for the global implication.

---

*Student-has-tell-tale-signs-of-knowledge - assume student has indirectly used the topic.
*Student-has-definitional-knowledge - assume student knows the definition of the topic.
*Student-has-background-information - assume student knows topic through prior experience.
*Student-is-actively-forming-knowledge - assume student is forming a model of the information.
*Student-is-confused - assume student is confused.
*Student-knows-the-topic - assume student has used topic correctly.
*Student-understands - assume student understands the topic.
*Student/domain-agreement - assume agreement between student's knowledge and domain know
*Student s-knowledge-threshold-known - assume student's threshold of knowledge is known.

Figure 13: Global implications about a student.

The student's third response is wrong and the tutor now is forced to reverse its current evaluation. After a single wrong answer, several immediate implications are available since they are bound to the conversational move: either the student does not know the material in question or he made a careless error. If we assume the former and recognize that the wrong answer came on the heels of two correct answers, we have a more complex implication: now it is possible to say that the topic might lie on the student's threshold of knowledge. This is because the student

***Topic-is-important** – assume topic is important.
***Topic-is-generally-known** – assume topic is generally known.
***Topic-is-learnable-elsewhere** – assume topic has been learned at another time.
***Topic-is-learnable-through-dialogue** – assume topic can be learned during the dialogue.
***Topic-is-on-student-threshold** – assume topic lies on the student threshold of knowledge.
***Topic-is-background-material** – assume topic was learned before the discourse.
***Topic-is-less-important** – assume topic is less important.
***Topic-is-irrelevant** – assume topic is irrelevant.
***Topic-was-complete** – assume topic was fully developed.
***Topic-has-been-popped** – assume topic lies at a higher level in knowledge base.

Figure 14: Global implications about topic.

knows some attributes about the topic, control variables, e.g., its definition and value before loop exit, yet he does not know at least one attribute, e.g., its value after loop exit.

The example shows how the system can infer more general knowledge about the student and the topic by using global implications. Since global interpretations are made over several interactions, additional evidence brought from earlier responses, can be weighed along with current implications to generate a more global view. In this way the system can achieve a broader view of student knowledge and topic complexity.

Additional global inferences that we expect the machine to make are presented in Figures 13 and 14. In each figure, global implications are listed on the left and the assessments of which they are a "gloss" are on the right.

5.5.3 Managing discourse using global implications

One of the primary objectives of this model is to use support for global implications to influence discourse behavior. Discourse management is handled by an ATN-like mechanism that allows both default and exceptional behavior.[10] Default behavior is based on traversal of the arcs of the ATN; exceptional behavior is achieved by activated meta-rules that move the system from one set of discourse states to a new set of states. Transitions within discourse states define the system's default behavior. For instance, the default response to a wrong answer might be a two state sequence: **explicitly acknowledge incorrect answer**, followed by **teach topic attribute**. This sequence can be abandoned if a meta-rule fires and replaces it with a sequence such as **provide example** and **question topic**. A meta-rule is a structure defined by preconditions, prior states, actions, and post-processing actions (see Appendix 1). Preconditions are largely built from global implications. Once a global implication passes threshold and triggers a meta-rule, the discourse manager will move to a new state sequence by a method described in detail by Woolf and McDonald[10].

Discourse behavior is determined by meta-rules, which in turn are enabled by global implications reaching threshold. Global implications will reach threshold as a result of support from the on-going discourse. The system supports global implications in a process that is analogous to the read-eval-print cycle of LISP. The top-level "thinking" of the machine is suggested in Figure 15. After a student responds, implications from his response will be placed on a stack and certain global implications will be activated. Global implications that gain support from the newly activated implications are endorsed i.e., given reasons to be believed or disbelieved.[21] The endorsements are associated with an applicability condition: "correct answer indicates correct information," is always possible when the response is correct;

STEP1: Tutor behaves according to default state sequence (consistent with current implications).
A) generate text
B) parse student's response

STEP2: Tutor identifies implications of student's utterance and endorses evidence for global implications

STEP3: Some endorsed global implications may reach threshold.

STEP4: Global implications at threshold may trigger meta-rules taking the system to a new state sequence.

STEP5: Go to 1.

Figure 15: Steps to manage discourse.

"correct answer indicates a guess" is applicable when the response is correct but earlier responses were wrong; and "could be a mistake" is applicable for any response. Global implications that pass beyond a threshold level are viable assessments of the topic or student; they can be used to activate changes in the system's discourse behavior. Some endorsements are *positive*, meaning they support the interpretation with which they are associated. Others are *negative*, meaning they provide reasons to disbelieve their associated interpretations.

In sum, the state of affairs of the discourse is given by support for or against global implications. When evidence for a change in interpretation exceeds threshold and the system has reason to endorse a new interpretation, it takes action and changes its teaching strategy. Customized tutoring behavior is achieved through recognition of the effects of these global implications.

5.6 Proposed Tutoring Discourse

As an example of the kind of high-performance tutoring system we intend to build using interpretation knowledge, we present a scenario of how our program would tutor a student in Pascal. Figure 16 shows the kind of problem students receive in our department's introductory Pascal course. Below the problem statement is a program actually written by a student.

PROBLEM: Write a program that finds the average grade for a student who types his grades in at the keyboard. After the last grade is typed in the student will type 9999. Please print out the average grade.

```
1 Program Student29 (input, output);
2 Var
3     sum, num, grade, ave : integer;
4 Begin
5     sum := 0;
6     num := 1;
7     read (grade);
8     while grade <> 9999 do
9       begin
10       read (grade);
11       sum := sum + grade;
12       num := num + 1
13      end;
14    while grade = 9999 do
15       begin
16       ave := sum/num;
17        writeln (ave)
18       end;
19 end.
```

Figure 16: A Student Program.

The program is syntactically correct but does not produce the desired result.[3] It reveals at least four underlying misconceptions about control variables, looping constructs, and flow of control. Cognitive studies of programming[13,22] suggest for instance, that the student may believe that:

>> the order of operations inside the loop should be READ/PROCESS (a Pascal WHILE loop is designed to be PROCESS/READ);

>> the value of a variable inside the loop is distinct from its value outside the loop;

>> the loop construct does not test the last value of the control variable; and

>> a single iteration of a sequence of commands requires a looping construct.

There is evidence for each misconception.[4] The task of the discourse that the tutor engages in is to resolve the ambiguity among the possible *misconceptions* and to focus the discourse on the repair of a single misconception. The discourse in Figure 17 is an example of what we expect our tutor to do and how it might unravel the student's misunderstandings and make him "verbalize" his malformed ideas. In order to generate that discourse, the tutor:

---

[3] The program is flawed in several ways. The value of the "read" command in line 7 will not be processed, because the "read" command in line 10 immediately replaces it with a new value. The second loop will never terminate because the control variable "grade" is never modified inside the loop body. We also suggest, that the student assumes that the second loop will execute only once since its only action is to print out the average. We call such a loop an "if-loop" and suggest that the student has confused the "if" conditional construct with the family of looping constructs.

[4] For instance, the fact that "read (grade)" in line 10 precedes the processing of grade in line 11 is evidence for the first misconception. In addition, the use of two loops, both controlled by the same variable, "grade," is evidence for the second and third misconceptions since they suggest that the student does not realize that the value of "grade" will be available at the end of the first loop and therefore does not have to be retested.

1. *Tutor*: Do you know that *GRADE* in line 8 is a control variable?
   Student: Yes

2. *Tutor*: O.K. What is the value of *GRADE* immediately before leaving the loop in line 13?
   Student: 9999

3. *Tutor*: Right, and what is its value immediately after leaving the WHILE loop, in line 13?
   Student: I don't know.

4. *Tutor*: O.K, suppose the values read in were 1 5 9 10 9999. How many times would your WHILE loop execute?
   Student: I don't know. Maybe 5 times.

5. *Tutor*: Well, what would be the value of *GRADE* immediately after leaving the loop in line 13?
   Student: I don't know.

6. *Tutor*: You seem to have a misconception. The loop would execute 4 times. Although 9999 would be read inside the loop during the fourth iteration and tested at the top of the fifth iteration, the entrance test for that iteration would fail since *GRADE* is now unequal to 9999. The fifth loop execution would never occur and control would pass to line 14 immediately after the loop.

   You probably assumed that the value of the variable inside the loop was different from its value outside the loop. In fact, whether inside or outside of the loop, variables in a WHILE loop have the same value. If *GRADE* equals 9999 before the loop terminates, it will still equal 9999 after the loop terminates. Testing the value of *GRADE* immediately after the WHILE loop is superfluous.

   Now, let me give you a new problem: Compute and print the average number of hours worked each day by a student employed during a month if hours per day is typed in and averaged by the program. Assume that hours per day are typed on a single line and followed by -1.

Figure 17: Proposed tutoring discourse for the Program in Figure 9.

---

focused on several topics in succession (e.g., control variables and while loops);

handled several errors (e.g., value the control variable and the number of loop iterations); and

customized new examples to present to the student (e.g., hypothetical program and a new problem).

Note that in Figure 17 the tutor asks one question (line 1) to establish that both it and the student share a common vocabulary about control variables. In the next two questions (lines 2-3) the tutor asks enough questions concerning misconceptions about variable values and control flow to establish that the student does, in fact believe that the value of *GRADES* is not available after the first loop terminates. In line 4 the tutor presents some example input custom-tailored to the problem and the student's history in order to verify its hypothesis that the student did not realize that the value of *GRADES* was available after the loop exited. Based on the student's response thus far the tutor (line 6) explains its diagnosis of the misconception in terms of characteristics of the presenting program: *GRADES* had a value of 9999 when the first loop terminated and after it terminated *GRADES* will retain that value. Thus, *GRADES* is available between the first and second loops.

5.7 Example Generation

Generating examples is a key feature of the proposed system and we will be working closely with Rissland[23] to enrich explanations with examples. Generating illuminating examples tailored to a student's level of knowledge requires knowing the student's activities, background and particularly his history of errors. Generation and modification of examples is a powerful technique both to refine the model of the student and as a tool for defining the student's level of understanding of the domain.[24] We propose to extend and apply previous work on *constrained example*

generation in which new examples are generated from old primarily by domain-specific modifications of existing examples. Some of these constraints are generated by general principles such as "Look at extreme cases," "Look at a simpler case." Other constraints will come from specific knowledge of an individual student, his context, past history, cognitive style, etc.

5.8 Summary

We have suggested a way to represent the implications and intentions of a speaker as distinct from representing actual utterances. In our model, a computer tutor makes inferences about a student's knowledge or domain topics based on constraints about the type of utterances spoken. Support for or against each implication is given by the type of conversational move. The tutor's control structure allows the systems to review or redirect the discourse based on the system's evaluation of implications it can make about the student's knowledge or the topic. We expect that evaluating implications will allow us to make predictions about managing subsequent discourses and judgements about the quality of the current discourse.

5.9 References

1. W. Mann, J. Moore, and J. Levin, "A comprehension model for human dialogue," *International Joint Conference on Artifical Intelligence*, 1977.

2. K. McKeown, "Generating relevant explanations: Natural language responses to questions about database structure," *National Proceedings of the Association of Artificial Intelligence*, 1980.

3. T. W. Finin, "Providing help and advice in task-oriented systems," *Proceedings IJCAI-83*, Karlsruhe, W. Germany, 1983.

4. R. Wilensky, "Talking to UNIX in english: An overview of UC," *Proceedings AAAI-82*, Pittsburgh, PA, August 1982.

5. W. Lehnert, M. Dyer, P. Johnson, C. Yang, and S. Harley, "Boris–An experiment in in-depth understanding of narratives," *Artificial Intelligence*, 20, pp. 15-62, 1983.

6. W. Lehnert and C. Loiselle, "An introduction to plot units," COINS Technical Report 84-13, Computer and Information Sciences, University of Massachusetts, Amherst, 1984.

7. D. McDonald and J. Pustejovsky, "TAGs as a grammatical formalism for generation," *Proceedings of the Association of Computational Linguistics*, 1985.

8. B. Woolf, *Context-dependent planning in a machine tutor*, Ph.D. Dissertation, Computer and Information Sciences, University of Massachusetts, Amherst, MA, 1984.

9. K. Ashley and E. Rissland, "Toward modelling legal arguments," *Proceedings of the Second International Congress, Logica, Informatica, Diritto, Automated Analysis of Legal Texts*, Florence, Italy, 1985.

10. B. Woolf and D. McDonald, "Design issues in building a computer tutor," in *IEEE Computer*, special issue on "Artificial Intelligence for Human-Machine Interaction" Sept 1984.

11. L. Johnson and E. M. Soloway, "PROUST: Knowledge-based program debugging," *Proceedings of the Eighth International Software Engineering Conference*, Orlando, FL, March 1984.

12. J. Bonar, "Collecting and analyzing on-line protocols from novice programmers," *Behavioral Research Methods and Instrumentation*, 1982a.

13. J. Bonar, "Natural problem solving strategies and programming language

constructs," *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, 1982b.

14. A. Collins, E. Warnock, and J. Passafiume, "Analysis and synthesis of tutorial dialogues," *Psychology of Learning and Motivation*, vol. 9, Academic Press, Inc., 1975.

15. R. Reichman, "Plain speaking: A theory and grammar of spontaneous discourse," Ph.D. thesis, Harvard University, Department of Mathematics, also Bolt, Beranek and Newman, Technical Report #4681, 1981.

16. L. Sass, "Parental communication deviance and schizophrenia: A cognitive-developmental analysis" in L. Vaina and H. Hintikka (eds.), *Cognitive Constraints on Communication*, 1984.

17. J. Larkin. J. McDermott, D. P. Simon, and H. Simon, "Expert and novice performance in solving physics problems," *Science*, Vol. 208, 20, 1980.

18. K. Forbus and A. Stevens, "Using qualitative simulation to generate explanations," Report #4480, Bolt, Beranek and Newman, Inc., 1981.

19. J. deKleer and J. S. Brown, "Foundations of envisioning," *Proceedings of the National Association of Artificial Intelligence*, 1982.

20. H. Grice, "Logic and conversation," in P. Cole and J. Morgan (eds.) *Syntax and Semantics*, Academic Press, New York, pp. 41-58, 1975.

21. M. Sullivan and P. Cohen, "An endorsement-based plan recognition program" *International Joint Conference on Artificial Intelligence*, Los Angeles, 1985.

22. E. Soloway, J. Bonar, B. Woolf, P. Barth, E. Rubin, and K. Erlich, "Cognition and programming: Why your students write those crazy programs," *Proceedings of the National Educational Computing Conference*, NECC, No. Denton, TX, 1981.

23. E. Rissland and E. Soloway, "Constrained example generation: A testbed for studying learning," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1981.

24. E. Rissland, "Understanding Mathematics," *Cognitive Science*, Vol. 2, No. 4, 1978.

5.10 Appendix

Meta-rules to generate exceptional discourse behavior

```
;*****************************************************************************
;
;  Global implications are written in bold font and prefaced with an asterisk (*)
;  Meta-rules are written in bold font
;  Discourse states are prefaced with a dollar sign ($)
;
;
;*****************************************************************************
make_$rule_struct 'PRESENT-EXAMPLE
      preconditions
                  (FC-and *found-threshold-of-knowledge
                  *teach-at-threshold-of-knowledge
                  *confused-student
                  *tutor-is-co-operative
                  *topic-is-important)
      action '(setg next_state '($present-example))
      post_processing '()
      prior_state '($question-model
                  $question-topic
                  $question-role-value)


make_$rule_struct 'PROBE-MISCONCEPTION
      preconditions
                  (FC-AND *evidence-of-misconception
                  *confused-student
                  *known-student-threshold)
      action '(setg next_state '$probe-misconception)
      post_processing '()
      prior_state '($teach)


make_$rule_struct  'JETTISON
      precondition *dialogue-is-ineffective
      action '(setg next_state (find-parent state))
      post-process '()
      prior_state '($ all states)


make_$rule_struct  'TEACH-TOPIC
      preconditions '(*topic-is-learnable-elsewhere)
                  not *known-student-threshold
```

```
                *direct-sign-of-student-knowledge)
action '(setg next_state '($teach-topic))
post_processing '()
prior_state '($explore-knowledge)
```

6. A knowledge-based approach to data management for intelligent user interfaces

A KNOWLEDGE-BASED APPROACH TO DATA MANAGEMENT FOR INTELLIGENT USER INTERFACES

Carol A. Broverman

W. Bruce Croft

Computer and Information Science Department
University of Massachusetts
Amherst, MA. 01003

ABSTRACT

An intelligent user interface (POISE) is described that provides facilities for defining and supporting higher-level user tasks. Although an object-based data model forms an important part of the POISE system, other types of knowledge such as task descriptions and tool descriptions are required. The management of instantiations of the task and object descriptions is a complex process because POISE both predicts user actions and allows multiple, competing interpretations of user actions. In this paper, we describe how the knowledge base (including the object data model) is defined and used by the intelligent interface. We also describe an implementation of the knowledge base in a frame-based representation language.

6.1 Introduction

The data models used in database systems provide languages for describing objects, relationships between objects, constraints, and actions that are to be performed on the objects [TSIC82]. A database management system manages the schemas defined with the data model and the instantiations of the schemas that result from dynamic processing (i.e. the database). The object-oriented information that is captured with a data model forms an important part of the knowledge required for an intelligent user interface, but additional mechanisms are required to describe and manage other types of knowledge that are essential for this application. In this paper, we shall show how data models and other types of knowledge can be combined and managed to support an intelligent user interface.

Certain types of information systems can be characterized as consisting of a set of tools that support user tasks in a particular environment. Office information systems are a good example of this type of system and they have been used as the major testbed for the intelligent interface described in this paper. The tools in current office systems are designed to carry out simple tasks that are common to most offices. For example, tasks such as communication, time management and document production are supported by the electronic mail, calendar and text editor tools. A more effective system would support higher-level tasks that are directly related to the goals or functions of the office. This type of task often involves decision-making, complex sequences of actions, and interaction with a number of other people. The intelligent interface described in this paper consists of formalisms used to describe tasks and mechanisms for managing instantiations of these tasks.

The main types of knowledge required for the intelligent user interface consist of task descriptions, object descriptions and tool descriptions. These descriptions are intimately related because tasks manipulate objects through the use of tools. For example, the task of processing orders in a particular company could be described in terms of actions of form and mail tools on various form objects, together with actions and decision points that have no corresponding tools. On the basis of this task description, the intelligent interface would, in effect, provide a "virtual" tool that could support the order processing task. By separating task descriptions from tool descriptions, the addition of new tools will affect only the way in which a task *is supported, rather than* the description of a task. This division between task and tool descriptions is analogous to the separation of logical and physical levels of description in data models.

Task descriptions in the intelligent user interface play a similar role to application programs in typical database systems in that they refer to objects defined in the knowledge base "schema." However, in contrast to the very structured nature of the algorithms specified in application programs, task descriptions often have steps that rely on the problem-solving abilities of the person(s) using the system [FIKE80,BARB83]. Task descriptions are constantly subject to change, both at an organizational level and by individuals. Task descriptions also represent only a typical way of carrying out a task and many exceptions are possible. The type of support provided by the intelligent interface depends on the amount of structure in the task involved. Generally, the interface can automate the more structured parts of a task and provide assistance to users for the less structured parts [CROF84].

In the next section, we give an overview of the POISE intelligent interface and its capabilities for task support. The third section contains a more detailed discussion of the types of knowledge used in POISE and the relationships between them. In particular, we discuss the role of a data model in the overall knowledge base and how constraints can be specified in both the object and task descriptions. During the operation of POISE a variety of task and object instantiations are created to record the current and predicted states of user activities. The management of these instantiations is the subject of the fourth section. Finally, we show how the knowledge base, including the data model, can be implemented using a frame-based representation language.

6.2 Overview of POISE

The POISE system provides task support on the basis of hierarchies of task descriptions. The task descriptions specify the typical steps involved in the task, the objects that are affected by the task, and the goals of the task steps. The ability to combine recognition of user actions and planning using the descriptions and goals gives POISE great flexibility in the type of task support it can provide.

A simplified diagram of the main POISE components is shown in Figure 1. The *knowledge base*, which is the main subject of this paper, consists of two main parts. The first part is the relatively static description of the tasks, objects, and tools in a particular environment. This part of the knowledge base also contains the dictionaries and other information used by the natural language analysis and generation components of the system.

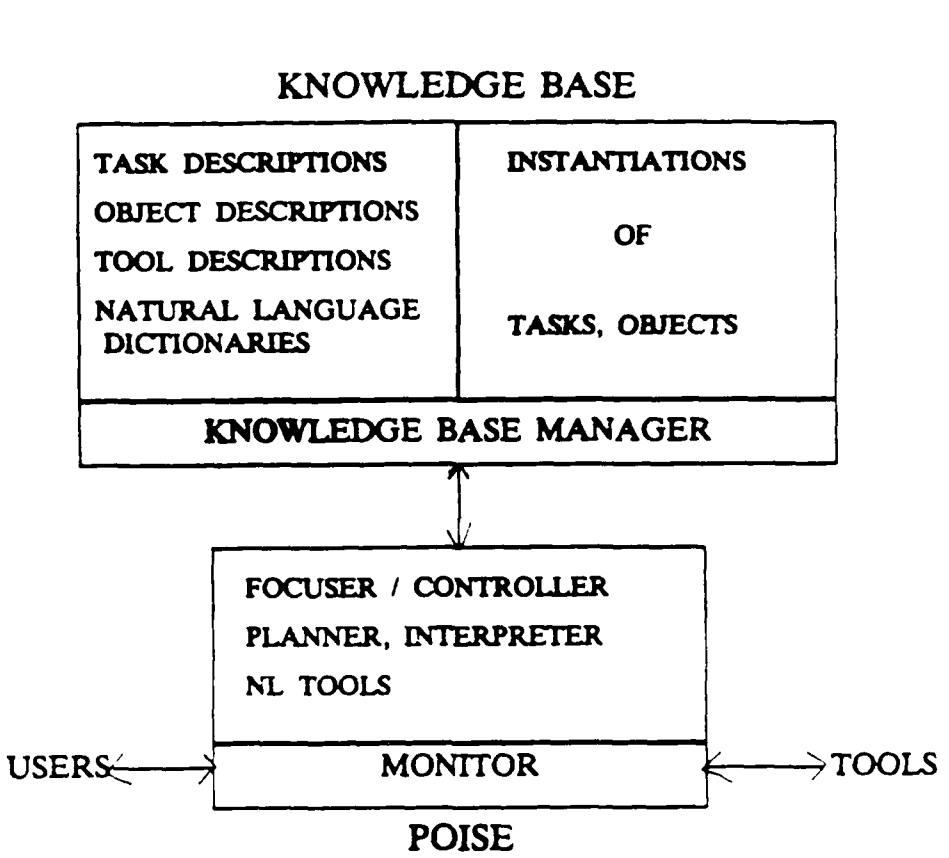


Figure 18: The POISE Intelligent Interface.

The second part of the knowledge base contains the instantiations which describe the dynamic state of the system. For example, in the static part of the knowledge base there might be a task description for filling out a purchase order form. There would also be a description of this form as an object type and its relationship to other form objects used by the system. After a user had started to fill in a particular purchase order form, the dynamic part of the knowledge base would contain a partial instantiation of the "Fill_out_purchase_order_form" task with values derived from the actual values filled in by the user. There would also be an instantiation of the database object that represents the actual purchase order form.

The instantiations in the knowledge base are generated and used by other POISE components such as the interpreter and planner. The knowledge base manager controls access to the knowledge base and provides the operations needed to

manipulate the knowledge.

The *monitor* provides the interface between the user, the tools and POISE. It is designed to allow the user to interact directly with "off-the-shelf" tools or to interact with tools through an interface specified within the monitor. This design avoids simulating within POISE the sophisticated interfaces of some tools, but enables the system to understand user actions. The tool descriptions, which are used by the monitor, define how tasks are implemented with the tools.

The other major components of POISE are the *focuser*, *interpreter* and *planner*. The interpreter is responsible for interpreting user actions in the context of the task descriptions. Since there may be multiple, concurrent, and competing interpretations of actions, the focuser is used to choose the most likely interpretations and to control the system's actions [CARV84]. The focuser must also provide a mechanism for backtracking should a user action or user error result in incorrect interpretations.

The planner is similar to the focuser in that it manages interpretations of user activities. However, in contrast to the focuser's emphasis on the recognition of user actions, the planner takes stated task goals and directs the user through sequences of actions designed to achieve those goals. The focuser, interpreter and planner must work in close cooperation for the system to be able to make predictions when attempting to recognize user actions, or to interpret user actions during the planner-directed execution of a task.

Both the interpreter and planner are aided by the constraint propagation which occurs during the execution of a task. Specific user actions apply constraints to the general task descriptions. Constraints also hold between steps specified in task descriptions and this information is used to propagate constraining parameters

throughout the executing task.

As stated previously, the main emphasis of the discussions in this paper is on the knowledge base and how it is used by POISE components such as the focuser, interpreter and planner.

6.3 The Knowledge Base

Tasks, objects and tools are represented in the knowledge base using different formalisms. These formalisms capture different, though related, information that is used by the intelligent interface. The following sections contain a discussion of each formalism and the type of information represented.

6.3.1 Task Descriptions

In order to represent the possible sequences of concurrent actions in a task, we are using a modified version of an Event Description Language [BATE84]. An example task description is presented in Figure 2. The algorithmic syntax of the procedure is specified by the IS clause, modified by the COND clause, and has its parameters defined by the WITH clause. The conditions required by a task in order to begin are specified by the PRECONDITION clause while the goals satisfied by a task are contained in the SATISFACTION clause.

The IS clause of the task definition provides a precise way of describing the standard algorithm for accomplishing a task in terms of other tasks and primitive operations (tool invocations). The sequence of constituent tasks is specified using regular expression operator, for example, Catenation (^) and Alternation (|).

```
PROC   Purchase_items

DESC   Procedure for puchasing items with non-state funds.

IS     Receive_purchase_request
       ' (Process_purchase_order | Process_purchase_requisition)
       ' Complete_purchase

WITH   *Items
       *Purchase_request = Receive_purchase_request.*Purchase_request
       *Purchase_order = Process_purchase_order.*Purchase_order_form
       *Invoice_mail_form = Complete_purchase.*Invoice_mail_form
       *Purchase_requisition =
       Process_purchase_requisition.*Purchase_requisition_form

COND   IF *Purchase_request/total_field <= 250
           THEN Process_purchase_order WILL_EXIST

       IF *Purchase_request/total_field > 250
           THEN Process_purchase_requisition WILL_EXIST

       FOR_PROPERTIES ("total_field" "itemized_order" "vendor_field")
                      (*Purchase_order MATCHES *Purchase_request  AND
                       *Purchase_order MATCHES *Invoice_mail_form)  OR
                      (*Purchase_requisition MATCHES *Purchase_request  AND
                       *Purchase_requisition MATCHES *Invoice_mail_form)

PRECONDITIONS —

SATISFACTION   *Items/status = delivered
```

Figure 19: An Example Procedure Specification.

The example shown in Figure 2 is a "Purchase_items" task. This task is a typical semi-structured clerical task. The IS clause of this task specifies that after a purchase request has been received, either a purchase requisition or a purchase order is processed. The task is completed by the steps involved in the Complete_purchase procedure. To get the details of the steps involved in the Complete_purchase task, we would have to examine the corresponding descriptions. The more detailed (or

less abstract) tasks contain links to the tools available in the system. The lowest level procedures in this hierarchy correspond (approximately) to tool invocations.

The attributes of a task are defined by the WITH clause. Task attributes may be constant values, associated objects, or attributes of constituent tasks. These attributes described by the WITH clause of a task may then be used by higher level subsuming tasks. In the example, the *Purchase_request object attribute is obtained from a parameter of the Receive_purchase_request subtask. Note that a reference to an object is marked by a preceding asterisk.

Constraints may be placed upon the values of task attributes and the relationships between task attributes. The COND clause is used to describe these constraints. In addition to rules specifying restrictions on constant values of attributes, the COND clause may also include rules establishing relationships between associated object and/or task attributes. For example, in Figure 2 we see that among other constraints, the *Purchase_requisition_form object used in the Process_purchase_requisition subtask must match the *Invoice_mail_form object used in the Complete_purchase subtask in terms of their total, itemized_order, and vendor fields. COND clause rules may also be used to distinguish between two tasks where only one of the two can occur (Alternation). In the example, the value of the "total_field" of the *Purchase_request attribute determines which of Process_purchase_order or Process_purchase_requisition can occur as a subtask.

Often a constraint (like the one just mentioned aboved) can be represented within an object description itself (e.g. a "*Purchase_order_form" must have its "total_field" less than or equal to $250.00) instead of in the relevant task's COND clause. However, additional power and clarity can result from the redundant

expression of the constraint in the task (as shown in Figure 2). Thus, when object-related constraints may be used to guide task choice, it is appropriate to represent them in both the task and object descriptions.

The COND and WITH clauses, along with the temporal ordering constraints found in the IS clause, describe the flow of objects between the subtasks of the higher-level task being described. The WITH clause depicts the vertical flow of objects through the task abstraction hierarchy.

The POISE formalism also contains a description of the state of the knowledge base that must exist in order for the task to begin. The PRECONDITION clause specifies this set of conditions.

Upon completion of a task, certain conditions must be satisfied. This information serves both as an aid to the planner and as an alternate means of recognizing the completion of a task. The SATISFACTION clause specifies these conditions on database state. The example task specifies that the items of concern in the procedure Purchase_items must have been delivered when the task was finished. Goal specification in terms of database state allows the interface to bypass the usual mode of plan interpretation, which is based on a strict algorithmic ordering of plan substeps. Each substep of the higher level plan (in this case Purchase_items) may also be characterized by its goals (which contribute toward the highest-level goal); together, the goals of the top-level plan and those of its substeps constitute an implicit goal hierarchy.

6.3.2 Object Descriptions

A data model for the office has been suggested by Gibbs and Tsichritzis [GIBB83,GIBB84]. It is an example of a semantic data model that was designed specifically for the office domain. We have adopted a subset of the features in this data model for the external representation of the objects in the POISE knowledge base. A summary of this subset follows.

An object type is the structural specification of a class of objects. The general form of an object type definition is:

define *object-type object-name*

begin

properties: {property definitions}

constituents: {*constituent definitions*}

mappings : {mapping definitions}

constraints: {constraint specifications}

end

The *properties* section defines the attributes possessed by that object. Properties of an object type may be hierarchically decomposed. For example, a vendor object may have an address as one of its properties. The address property may be decomposed into street and city, street may be further decomposed into number and street name, etc. Each successive decomposition would be listed in the properties section of the object. Properties that are multi-valued are also easily specified.

The *constituents* section describes an object in terms of constituent objects rather than properties. The *mappings* section defines relationships among the constituents specified in the *constituents* section. Therefore, a *mappings* section will

only be present when a *constituents* section is present.

The *constraints* section is used to specify data type constraints on properties, object type constraints on constituents and uniqueness constraints on properties and/or constituents.

Specialization relationships are defined via a special declaration of the form *Object2 ISA Object1*, where Object1 and Object2 are declared object names. This declaration implies inheritance from Object1 to Object2 of all Object1's properties, constituents, mappings and constraints.

In addition to *object types*, *domains* and *triggers* may be specified. *Domains* are abstract datatypes which are used to define non-primitive data types. In the office automation application, for example, *dates* may be a domain and its associated functions would parse date specifications, modify them, display them, compute time differences, etc. *Triggers* are demon-like entities that carry out specified actions when certain conditions are met upon the invocation of specified database operations.

The use of domains has been presented as a declarative method for representing constraints that are imposed on single properties. Triggers are procedural rather than declarative, and they represent object-related constraints dealing with relationships between different parts of a single object. Triggers are used for constraints that hold between one or more slots of a single object, or for other such constraints which are more complex than simple data type specifications. They can also be used to implement simple tasks, such as sending mail out on a specified schedule. The POISE system treats tasks implemented this way as part of the available tool set.

The object data model provides operations to add and remove object instances in the knowledge base, to modify object instances, to retrieve object instances and to define transactions. The use of transactions in a task-oriented environment such as POISE raises some interesting questions. Tasks such as filling out a form could be specified as transactions but, unlike the execution of an application program, a task can be suspended indefinitely at any point by the user. The locks associated with these suspended transactions could cause significant problems. Another problem is that POISE provides assistance by performing constraint checking as soon as possible whereas the normal definition of transactions can involve more than one user action and allows constraint violation during the transaction. These issues are being explored in the current application.

It may be possible to represent task descriptions as objects using the model just presented. Substeps can be listed in the *constituent* section of the specification, WITH clause attributes can be represented in the *property* section, and the COND constraints can be represented either in the *constraint* section or as *triggers*. The formalism would require extensions to allow for the representation of the PRECONDITION and SATISFACTION clauses of the EDL task descriptions.

6.3.3 Tool Descriptions

The monitor acts as the POISE interface to the external world by performing mappings between events in the user and task domains and events in the lowest level of POISE task descriptions. It recognizes user actions on behalf of the planning and interpretation components and performs actions at the request of the planner. The mappings between tool functionality, user actions and task descriptions constitute the

tool descriptions that are defined whenever a new tool is added to the system.

The tool mappings can be quite complex; one can usually access a tool function only through a pre-defined interface that was designed for human users. A major part of the tool mappings are the functions that access tool functionality through the tool's existing interface. The use of access functions implies that changes to the tool set need affect only the tool mappings in the monitor, and do not affect the POISE task descriptions.

A set of mappings is also defined between objects that tools know about and objects in the POISE database. Tools manipulate objects and POISE maintains instantiations of selected objects. If an object changed by a tool is stored in the POISE knowledge base, the monitor informs POISE of the change. Note that these mappings would not be necessary if the tool set and the intelligent interface were designed as an integrated package. For example, the forms tool in the current POISE system is implemented directly using forms and operators defined by the object data model.

The lowest level POISE task descriptions, called primitives, have an important role in the tool descriptions. The primitives define what user actions POISE will be interested in, and what tool functionality will need to be accessed by POISE. Consequently, for each action corresponding to a POISE primitive, the capability must exist for both recognizing and performing that action through the tool interface. The POISE primitives thus effectively describe the granularity of the tool descriptions.

6.4 Instantiation Management

The POISE *interface* is capable of running in two different modes: interpretation and planning. While in interpretation mode (the start-up and default mode in the current application), POISE maintains consistent interpretations or integrated views of user actions as they are being performed. These interpretations are represented by hierarchical structures of instantiations of static task templates. That is, when a user action is recognized by the interface, a structure is created which is based on the appropriate static task template; this structure is then incorporated into a hierarchical interpretation. The instantiation will contain the dynamically determined parameter values particular to that user action, as well as the static constraints found in the template.

The dynamic portion of the knowledge base contains all current interpretations of user actions; some of which may have been designated as more likely than others by the focuser as the interpretation process proceeds. In addition, as each new step is taken by the user, POISE retains a copy of the previous interpretation in order to facilitate backtracking in the event of an interpretation error.

The second function or mode of the POISE *interface* is to provide the automation of user tasks through the use of the planning subsystem. The user is able to request the system to carry out a task or a series of tasks by specifying a higher-level POISE task (task-oriented planning) or by indicating conditions which must exist in the dynamic knowledge base that would represent the completion of the targeted task (goal-oriented planning). In the first case, the user may invoke the planning subsystem to complete an existing (partially instantiated) task or to carry

out a new task. In the latter case, the intelligent interface is responsible for determining the most appropriate way (e.g. sequence of POISE tasks) of achieving the specified goals. For example, the user may explicitly request the interface to complete the task of purchasing a desk once a purchase request has been received (task-oriented planning). The user could also specify the acquisition of a desk as a goal, and the system could determine how to achieve that goal (e.g. through an instantiation of the Purchase_Items task).

An example snapshot view of the dynamic knowledge base (Figure 3) illustrates some characteristics of procedure and object instantiation management. As procedure instantiations are created, object instantiations are also created to represent the associated database objects. These object instantiations are created by the POISE monitor when a tool creates an object in the application domain. The object instantiations created in this way are known as "base objects," and they uniquely correspond to the real objects being manipulated by the user. "Base-object" instantiations are linked to the task descriptions which refer to them as well as to related object instantiations. For example, a base-object may be connected to another base-object which contains it.

In the example, we can see that when the user invokes the mail tool to read his/her mail, the interface creates an instantiation of the Receive_information procedure, and POISE also creates associated object instantiations to represent the Purchase_request_form and the Purchase_request_mail_object manipulated by the tool. The database status of these objects indicates that they are base objects. These two objects are linked together to show that the Purchase_request_form is part of the Purchase_request_mail_object.

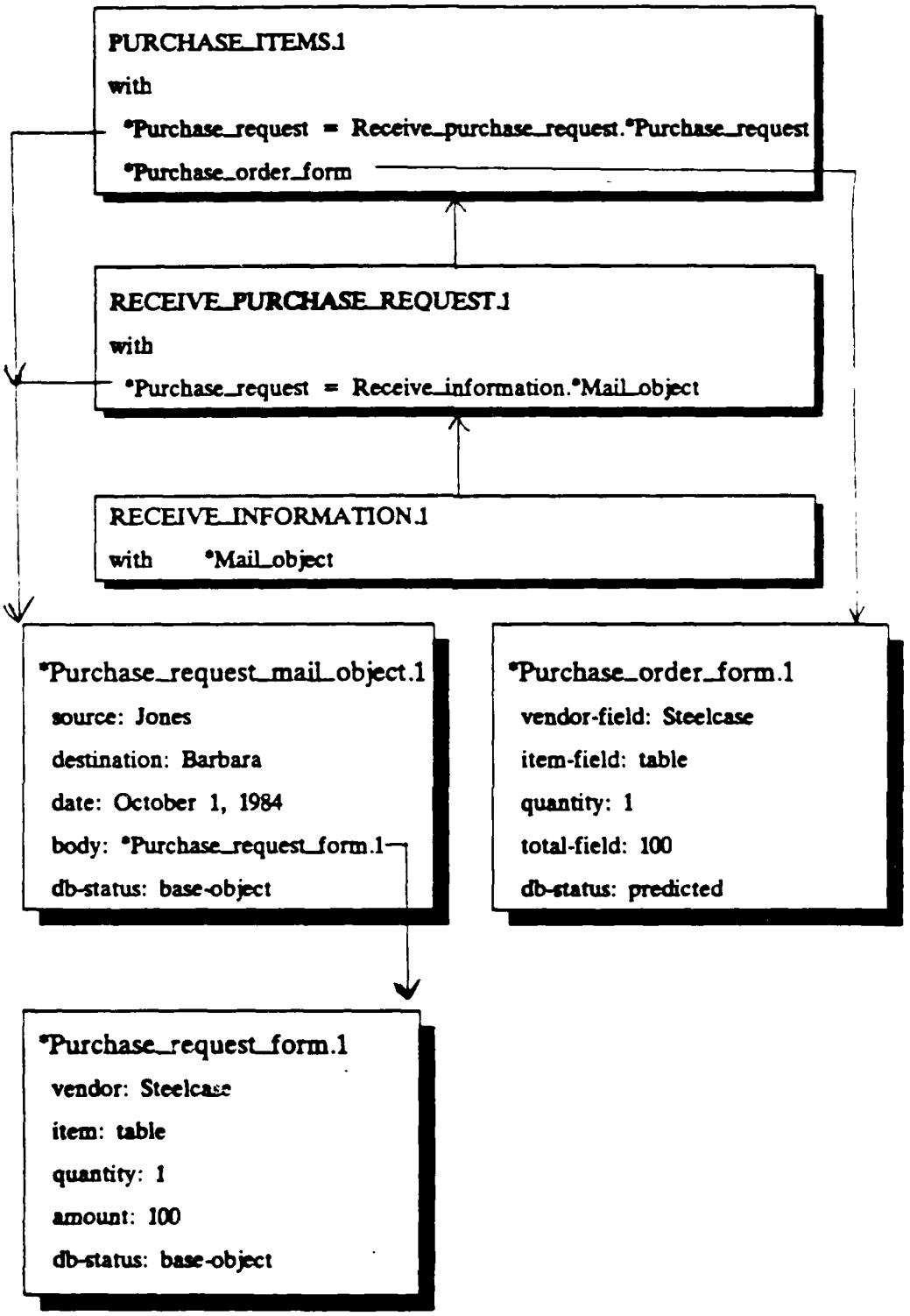


Figure 20: Simplified example of instantiations during task execution.

A second type of object instantiation (in addition to "base-objects") are "predicted" object instantiations. Predicted objects are represented in much the same way as base objects, but they are conceptually different. Predicted objects serve as placeholders for constraints associated with an interpretation, and do not correspond to an object that the user (or tool) has actually created or manipulated. The constraints embodied by the predicted objects are derived from the descriptions of tasks making up the interpretation as well as from the object descriptions associated with the interpretation. The use of predicted objects provides additional guidance during the parsing of user actions into consistent and unambiguous interpretations, and facilitates the propagation of constraints among related object instantiations.

Again looking at the example in Figure 3, we see that as interpretation proceeds by abstracting unambiguously *up to* Receive_purchase_request and Purchase_items, the object associated with the new instantiation of Purchase_items is represented by a predicted instantiation, since this object has not actually been created by a tool. However, the predictions of this object and the constraints it embodies are recorded in the dynamic knowledge base.

In contrast to traditional database management, where the requirement is to maintain object instantiations, the maintenance of a knowledge base for an intelligent user interface involves multi-leveled demands. These demands include the management of the base object instantiations, predicted objects, task instantiations, and the relationships between all of these different types of instantiations. Also, interpretations of instantiations exist as another type of unit to maintain, and the knowledge base manager must handle multiple copies of possibly conflicting interpretations. Many of these management requirements can be handled by

representing additional information in the object data model schemas. For example, a base object may have defined relationships to all predicted objects which may constrain it, and triggers may be attached to propagate changes directly affecting a base object to its associated predicted objects for consistency checking.

6.5 A Frame-Based Implementation

Frame-based languages [BARR81] and other artificial intelligence representation tools are appropriate choices for implementing the knowledge base described thus far. These knowledge representation techniques offer a skeletal structure which is much less restrictive than traditional data models, and offer the needed flexibility for representing a broad variety of types of knowledge and their associated constraints. While a frame-based representation language can be used to represent the entire knowledge base, only the data model is currently implemented with such a tool, while the other sections are under development. The data model's view of an object as an aggregate of properties or constituents is easily mapped to a frame-based view where a frame represents a concept as an aggregate of its slots. The frame-based implementation of the object portion of the knowledge base is described in the remainder of this section.

Object descriptions specified in the external language of the data model are converted by POISE to the chosen internal representation, which is SRL (Schema Representation Language) [WRIG83]. SRL is a frame-based language that has been a testing ground for exploring issues in inheritance. SRL was designed to allow maximum flexibility in definition of the desired representation.

Some of the features that SRL provides are: multiple roles (and thereby multiple inheritance paths), multiple contexts, facilities for the specification of default values, a search path specification language, demons, user-defined relations as well as useful system-defined relations, and various accompanying packages such as a query interface and some database utility functions. In the development of SRL, special attention has been paid to dealing with the problems of differentiating multiple-path inheritance, allowing users to define their own relations and inheritance semantics, and allowing selective search specifications made by the user.

In our implementation of the knowledge base in SRL, we have used the IS-A link to define simple inheritance paths. Objects are implemented as "schemas" in SRL, with the "slots" of the schemas corresponding to what the data model refers to as object constituents and properties.

Constraints on property values are specified both declaratively and procedurally, following the distinction outlined in the data model. Single-property constraints are specified declaratively using domains. Domain definitions are embodied in a special type of object. More complex constraints relating more than one property of an object are expressed procedurally using a trigger mechanism (called "demons" in SRL).

Simple data-type restrictions are specified by an "range" attachment to the slot of interest, and SRL has a built-in mechanism for executing the procedural check corresponding to the declarative form whenever a value is put into that slot. More complex data-checking, specifically that involving inter-slot dependencies, is performed via a procedural demon mechanism. The triggers in the formal model are implemented using SRL's demon facility. Demons are attached to the slots of

concern, and fire when triggered by a specified type of access (determined by the system designer). Demons are used in the current application for both complex value-checking and automatic slot-filling when possible.

SRL offers rather complex facilities for specialized inheritance (other than the traditional IS-A link) and additional type of inheritance links, which for the most part were judged to be too complex for the present application. An additional link which is used is the *instance* link, which is similar to the *is-a* link in terms of inheritance, but allows one to distinguish between the concepts in the hierarchy and the actual world objects corresponding to instantiations of concepts.

6.6 Summary

An intelligent user interface that can both recognize and carry out user tasks requires complex knowledge representation and management techniques. A typical database system can only partially fulfill those requirements. In particular, a database system does not provide facilities for defining semi-structured tasks and managing the instantiations of those tasks that arise from different interpretations and predictions of user actions. The major components of the intelligent user interface knowledge base (task descriptions, object descriptions and tool descriptions) can, however, be regarded as extensions of components of database systems (application programs and schemas).

In the system described in the paper, formalisms for defining tasks and tools are combined with a data model for describing the objects they manipulate. Constraints can be related both to objects and to the way in which objects are used by tasks. Instantiations of the tasks and associated objects are created and used by

the focuser, planner and interpreter modules of the POISE system. A more integrated and uniform formalism for the representation of objects, tasks and tools may be appropriate, and is currently under investigation.

Acknowledgements

The POISE system was developed with V. Lesser, N. Carver, A. Hough, and L. Lefkowitz. This research was funded in part by grants from the Digital Equipment Corporation External Research Program and the Rome Air Development Center (RADC).

6.7 References

BARR81 Barr, Avron; Feigenbaum, Edward A.; eds. *The Handbook of Artificial Intelligence*, Vol. 2; 1981.

BARB83 Barber, G. "Supporting organizational problem solving with a work station". *ACM Transactions on Office Information Systems*, 1: 45-67; 1983.

BATE84 Bates, P.C.; Wileden, J.C. "High-level debugging of distributed systems: The behavioral abstraction approach". *Journal of Systems and Software*, 3: 255-264; 1984.

CARV84 Carver, Norman F.; Lesser, Victor R.; McCue, Daniel L. "Focusing in Plan Recognition". *Proceedings of the National Conference on Artificial Intelligence*; Austin, Texas, 1984.

CROF84 Croft, W. Bruce; Lefkowitz, Lawrence S. "Task Support in an Office System". *ACM Transactions of Office Information Systems*, 2: 197-212; 1984.

FIKE80 Fikes, R.E.; Henderson, D.A. "On supporting the use of procedures in office work". *First National Conference on Artificial Intelligence*, Stanford, California, 1980.

GIBB84 Gibbs, S. "An object-oriented office data model." Ph.D. Thesis, University of Toronto, Canada, 1984.

GIBB83 Gibbs, S.; Tsichritzis, D. "A data modeling approach for office information systems", *ACM Transactions on Office Information Systems*, 1: 299-319; 1983.

TSIC82 Tsichritzis, D.; Lochovsky, F. *Data Models*. Prentice-Hall, 1982.

WRIG83 Wright, M.; Fox, M.S. *SRL 1.5 User Manual*, Intelligent Systems Laboratory, Carnegie-Mellon University Robotics Institute, 1983.

7. POISE Graphical Interface for Task Specification

7.1 Introduction and Objectives

The POISE graphical interface is designed to simplify a user's task of specifying procedures. A "procedure" is a set of ordered tasks that constitute a higher level task. The interface addresses the general issue of how to specify processes; it is not an interface for a particular domain of procedures. The major advantages of using a graphical interface are its (1) suitability for temporal ordering, (2) ease of learning and use by naive users, and (3) adaptability to different procedure domains.

The interface satisfies several long-term user needs. First of all, it provides an editor for the creation, modification and saving of procedure specifications. Secondly, it will eventually include a Procedure Library, which will provide an overview and index of procedures' calling relationships and structure. Thirdly, it will be capable of translating the graphical representation into sophisticated Event Description Language expressions. EDL is the system's internal representation of procedure specifications, which have previously been entered into the system via a LISP programmer. Finally, the interface will eventually allow the EDL files to be retrieved and converted to graphical representations for the users' viewing.

These features are dependent upon several system features that are described below. We begin with a top-down approach to procedure specification and assume that libraries of information will be accessed during the users' interactions with the interface. These libraries would include: (1) object descriptions (2) system tools

available for invocation, (3) user-defined tools, and (4) specified procedures. Pop-up menus appropriate for selecting interface options and for carrying out the specifications will provide the user with easy access to the system. Simple error-checking will ensure that the user doesn't inadvertently create procedures that violate certain constraints. Error messages, prompts, and status information about current and previous contexts/operations will further guide the user. A mouse will be used as a selection switch.

Finally, pilot-testing will be used to indicate whether our choice of graphical representations are clear and appropriate for naive users. We will also ascertain how to best expand the interface capabilities to allow for complete procedure specification.

7.2 Rationale for Implementation Decisions

7.2.1 Underlying Knowledge Representation

The key issue concerning knowledge representation was to determine which frame-based system would be most appropriate for this application. The choices included the simple defstruct-based frames, a more object-oriented flavors-based system, or frame packages, either commercial or COINS-grown. This application needs frames to represent relatively simple objects. In addition, because a concise internal representation could record the task specifications without saving the frame arrays themselves, there was no need for frame cataloguing capabilities. These factors led us to select the defstruct option, which is straightforward and convenient.

The goal_frames are linked hierarchically by means of the operands slot. That is, the elements of the operands slot of a parent frame are the names of the children frames of the parent frame. The name and icon slots are useful for referring to and labelling the frames and their graphical representations. Finally, the X-offset, Y-offset, width and height slots provide physical information about where the graphical objects are to appear on the screen.

Because frames cannot be stored and accessed after the machine has been cold-booted, and also because we were striving to generate EDL, an EDL LISP form seemed to be the most appropriate form in which to store the task-specification information. EDL can represent a complete as well as an incomplete specification. Given A followed by (B shuffle C), the EDL postfix form is: (A ! (B S C)) where the ! and S stand for concatenation and *shuffle*, respectively.

Examination of the EDL grammar is necessary to check for errors that a naive user might easily create. For example, in the DELETE-A-STEP function, there is a check to ensure that the deleted step did not leave a shuffle or an alternate frame with only one component operand. This check is based on the syntactical rule that shuffles and alternates must consist of two or more distinct components. The multiple and optional occurrence relations also provide a place to apply syntactical rules. For example, if the user applied the OPTIONAL-RELATION (0 or 1) and then applied the PLUS-RELATION (>= 1) to an object, then the system should recognize that the STAR-RELATION (>= 0) is actually in effect. This transitivity effect is not implemented. See the Figure 21 for a clear explanation of the frame slot and EDL conventions:

Operator Meaning/Name	Operator Code	Graphical Rep.	EDL Symbol
Sequential = Concatenation	!		!
Order Unimportant = Shuffle	S		#
One or the other = Alternate	A		|
Primitive	P		task icon

Number of Occurrences of a given step (within its temporal position):

Optional	(0 or 1)	opt-flag = T	{}
Plus	(>= 1)	plus-flag = T	+
Star	(>= 0)	star-flag = T	*

Anatomy of a typical frame:

Name	: name used to reference frame
Operator	: relationship among operands
Operands	: parameters of the operator—i.e. children
Opt-flag	: default value is nil
Plus-flag	: changed to T(rue) when # occurrences of
Star-flag	: a step is modified
Icon	: abbreviated label for graphical representation
Width	: based on fixed size primitives
Height	: incorporates border offsets in case of bounding box
X-offset	: Absolute offset from left-most graphical object. This also incorporates width of concatenation arrows. The offset is to the X Y coordinates
Y-offset	: of each graphical object's upper lefthand corner.

Figure 21: Frame Slot and EDL conventions.

7.2.2 Graphical Interface Considerations

Several considerations were incorporated into the graphical interface. First of all, the icons themselves were simplified visually from a previous implementation of the system. Our intention was to make the icons easy to learn and remember. While concatenation is intuitive, some examples might be helpful for the shuffle and alternate relations. Another concern was that the interactive nature of the menus, mouse-clicks and keyboard entries would facilitate task specification for the naive

user. Such activities require the ability to undo actions and to check for user errors. Finally, we aimed to create a layered graphical interface. This was advantageous because of the many input/output difficulties we encountered with the LISP-listener panes and with the permanent command menu. However, after we created the underlying functionality of the interface, it was not so easy to attach menus and mouse click methods.

7.2.3 Machine and Language Used

Because of the transitional nature of the project, we worked in ZetaLISP on the Symbolics LISP Machine. Due to machine availability, we later switched to the TI Explorer. The rest of the POISE system is in the process of being rewritten in Common LISP to be moved to the Symbolics LISP Machine.

7.3 Frame Tools, Utilities, and Demonstrations

7.3.1 Frame Tools

Frame tools involve getting and/or setting frame slot values. Sometimes this involves changing a given slot value by a given offset. Other times a predicate function checks for a certain slot value and returns T or nil. Sometimes "linked-frame-shifting" is desired to reset the X-offsets after a frame has been added, deleted, or combined with other frames. We include some miscellaneous LISP functions here that manipulate the value of the operands slot, for instance, without permanently changing it. This is specifically used to make sublists that require X-offset shifting.

We chose to set the prefix of all the gensymed symbol names to TEMP. This proves valuable in NESTP, the function that tests whether a frame is a gensymed result.

Finally, there are several versions of similar functions and macros that remain available for future use.

7.3.2 Utilities

(1) Frames created by a defstruct can be viewed by the following:

(describe-defstruct actual-frame-name 'frame-definition-name)

e.g. (describe-defstruct purchase-item 'goal_frame)

(2) (SHOWFR purchase_item) essentially calls the describe-defstruct for goal_frame type frames.

(3) (SEETREE purchase_item) displays the purchase_item frame and then each of the frames that are named in purchase_item's operands list.

(4) (Break "You are here test variables or press resume to continue.")

(5) (SETXY 115 234) sets the global *mouse-x* and *mouse-y* variables, respectively.

(6) (TELLXY) prints out the current values of the global variables, *mouse-x* and *mouse-y*. SETXY and TELLXY can be used conveniently to check the WHEREXY file's functions.

The WHEREXY file's functions determine the most specific graphic object that contains the given X,Y position. During development, we set and checked the X and Y positions via SETXY and TELLXY. In the future, they will be set by a mouse-clicks defmethod that affects the *work-window* pane, i.e., X and Y coordinates, which will be grabbed from the mouse when it is clicked, will be the input parameters for the INNER-BOX function in the WHEREXY file.

We designed an "opportunistic" search strategy for the job of finding the innermost box. The system can successfully recognize an X,Y location that falls within a top-level primitive, compound, or nested box. This search strategy is opportunistic, or best-first, in the sense that the next most probable box is probed and back-tracking is done whenever the next box is found to be a dead end.

7.3.3 Demonstrations

Demonstrations can occur at different levels of the interaction. One always has the option to call the primitive LISP functions, or at the higher end, one may simply call (INTERACT) and the entire task-query->display->modification and redraw loop->wrapup with EDL cycle will be conducted. At present, the MOD-QUERY and WRAPUP functions need to be completed. In addition, there is an I/O problem with LISP-listener in the graphics editor and LISP-listener window on the Symbolics. This problem can be circumvented by the protocol below. Furthermore, the protocal exhibits all the functionality of the project.

(1) In a LISP-listener, type: (load "hillary:servi;initfile.LISP")

to load in the eight program files.

(2) (make-gred) -> creates and exposes the graphics editor

***From now on, the function call must be done from the LISP listener pane of the graphics editor. ***

(3) (task-query) -> prompts user for original task specification

(4) (redraw) -> clears graphics pane and draws current task

(5) enter any low-level modification function

(6) (redraw) -> clears graphics pane and redraws updated task

(7) (wrap-up) -> generates EDL for current task

**steps 5 and 6 may be repeated as many times as desired

Presently, typing (mod-query) will cause the modifications menu to pop up. The program will await a mouse selection from this menu, even though it will not do anything about the chosen entry.

An alternate demonstration is available by typing (TESTIT), a canned sequence of specifications and modifications. One might need to setq *top-frame* to purchase_item in order run the display functions.

7.4 Future Work

The first area for expansion should be creating and making accessible a set of tasks from a Procedure Library. This might allow for selection and graphical display, incorporation of a task as a subtask of a current task specification, and exploration of hierarchical relationships among tasks. One might also tie constraints to the frame structure.

It would be desirable to have the capability to interrupt specification of one task to specify another task, which then is, in itself, a subtask of the first task (i.e., depth-1st and breadth-1st intermingled). This would undoubtedly require pop-up windows for separate displays.

Another area of future research should be how one associates meaning with the icons. Some example specifications and modifications might be helpful in addition to basic descriptions of the modification operators. Finally, we think it would be ideal to include icons on the modification operators menu so that the user has an easy reference to the icon-concept pair.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability, maintainability, and compatibility.

END

DATE

FILMED

APRIL

1988

DTIC