

DTIC FILE COPY

(4)

RADC-TR-88-324, Vol VII (of nine), Part A
Interim Report
March 1989

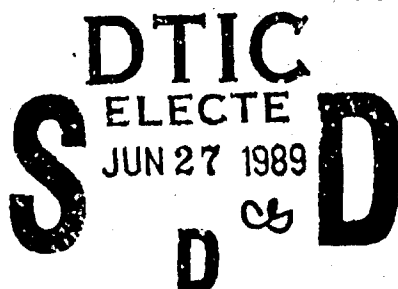
AD-A210 331



NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Time Oriented Problem Solving

Syracuse University

James F. Allen



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

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

89 6 27 058

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

RADC-TR-88-324, Vol VII (of nine), Part A has been reviewed and is approved for publication.

APPROVED:



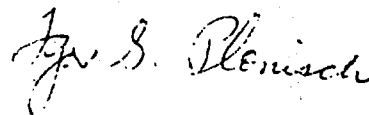
NORTHROP FOWLER, III
Project Engineer.

APPROVED:



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

FOR THE COMMANDER:



IGOR G. PLONISCH
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				Form Approved OMB No. 0704-0188	
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-88-324, Vol VII (of nine), Part A		
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC)		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)		
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240			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		PROJECT NO. 5581	TASK NO. 27	WORK UNIT ACCESSION NO. 13	
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Time Oriented Problem Solving					
12. PERSONAL AUTHOR(S) James F. Allen					
13a. TYPE OF REPORT Interim		13b. TIME COVERED FROM Dec 86 TO Dec 87		14. DATE OF REPORT (Year, Month, Day) March 1989	
15. PAGE COUNT 338					
16. SUPPLEMENTARY NOTATION This effort was performed as a subcontract by the University of Rochester to Syracuse University, Office of Sponsored Programs.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Temporal Reasoning, Planning, Plan Recognition		
12	05				
12	07				
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 third year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, dis- tributed AI for communications system control, automatic photo interpretation, time oriented problem solving, speech understanding systems, knowledge base maintenance, hardware archi- tectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system. The specific topic for this volume is a model theory and axiomatization of a logic for reasoning about planning in domains of con- current actions.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL NORTHROP FOWLER, III			22b. TELEPHONE (Include Area Code) (315) 330-7794		22c. OFFICE SYMBOL RADC/COES

DD Form 1473, JUN 86

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

7A Task A: Time Oriented Problem Solving

Report submitted by:

James F. Allen

Computer Science Department
University of Rochester
Rochester, NY

TABLE OF CONTENTS

7A.1	Executive Summary	7A-2
7A.2	Introduction	7A-3
7A.3	Time-Oriented Problem Solving	
7A.3.1	Time Points and Moments	7A-4
7A.3.2	A Formal Theory of Plan Recognition	7A-4
7A.3.3	Abstraction in Planning	7A-5
7A.3.4	Categories and Planning	7A-7
7A.4	Rhetorical	7A-8
	Publication List	7A-10
Appendix 7A-A	A Common-Sense Theory of Time	7A-A-1
Appendix 7A-B	Performance in Practical Problem Solving	7A-B-1
Appendix 7A-C	Short Time Periods	7A-C-1
Appendix 7A-D	A Formal Theory of Plan Recognition	7A-D-1
Appendix 7A-E	A Model of Concurrent Actions Having Temporal Extent ..	7A-E-1
Appendix 7A-F	Preserving Consistency Across Abstraction Mappings	7A-F-1



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

7A.1 Executive Summary

In the past year we have made progress in several areas relating to problem solving in temporally complex domains. The principle results are the following:

- an extension of the interval-based theory of time (developed in previous years under RADC funding) to allow two different forms of time-points;
- a generalized model of plan recognition, both as a formal theory and as a family of practical recognition algorithms (Henry Kautz, Ph.D., 1987);
- the formulation of two distinct forms of abstraction useful for planning and the investigation of their properties (Tenenbourg, 1987);
- an extension to the STRIPS representation of action that allows indirect effects and effects that are dependent on context (Weber, to appear);
- the development of an initial version of the RHET representation system, extending the HORNE system (see Miller & Allen, 1987).

Each of these projects is described in more detail below. Another important event this year is that the HORNE system, completed in 1986, has been distributed to over 50 different research sites in the last year, including places such as Advanced Decision Systems, SRI International and MCC.

In the project on formal temporal models, we axiomatized a theory of time in terms of intervals and the single relation MEET and showed that this theory subsumes Allen's interval-based theory. We then extended the theory by formally defining the beginnings and endings of intervals and showed that these have properties we normally would associate with points. We distinguished between these point-like objects and the concept of *moment* as hypothesized in discrete time models. Finally, we examined the theory in terms of each of several different models.

Henry Kautz completed his Ph.D. thesis which involved the first formal description of the plan recognition process and completed an initial implementation of a plan recognizer that uses action abstraction to reduce the combinatorial explosion found in simple plan recognition algorithms. The result is a family of plan recognition algorithms, each of increasing complexity and cost. Most notable is this system's ability to handle observed actions that are actually steps in two plans being executed simultaneously, actions that are only vaguely specified, and the ability to accept observed actions in any temporal ordering.

We have completed the first phase of the implementation of the RHET system, a knowledge representation language based on our previous representation system, HORNE. This work included design of the basic system including all the features of HORNE extended with a version of conniver-like contexts, full equality reasoning in contexts, a TMS and an extended type reasoning system. This system will be tested in some realistic application in natural language and planning applications in early '88.

We also explored different uses of action abstraction in planning systems. Josh Tenenbourg's Ph.D. thesis, to be completed in early 1988, defines two forms of abstraction: one found by simply relaxing the conditions under which actions may

apply, and the second by using a predicate mapping framework to define analogies between actions. With the first, he developed a theory in which it can be shown that if a fully specific plan exists for some problem, then there exists an abstract solution to an abstraction of the problem. While this might seem an obvious property, many previous models of abstraction do not guarantee this. With the second form, based on analogy, the opposite property can be proven: an abstract solution to a problem guarantees the existence of a fully concrete solution.

The final area concerns the representation and use of actions when reasoning about in a complex world. Our approach, based on the notion of action generation, has important improvements over the pervasive STRIPS representation of action as well as recent more formal theories, including the treatment of indirect effects, low-level actions, and action co-occurrence. The preliminary results are encouraging and this project will be completed as part of Jay Weber's Ph.D. dissertation (expected early 1989). Future work will include a more general representation of action and time that allows actions to have durations, thereby allowing serial action composition and requiring the generalization of preconditions to include properties that must hold during the action.

7A.2 Introduction

The unifying theme underlying all of the sub projects is that of producing new knowledge representation formalisms that extend the range of situations in which problem solving systems can be applied. The primary extension examined so far has been the use of temporal knowledge. By introducing time explicitly into the representation, a wide range of new situations can be represented including ones where more than one action may occur at a time and where external events may occur beyond the control of the planner.

In addition, using time explicitly in the representation allows for a variety of formal models of action that allow the first precisely defined model of the plan recognition process, as well as a formal definition of what it means for one action to be an abstraction of another. With these advances, confusions made in earlier planning systems can be identified and corrected, and more general systems can be developed.

In the rest of this report we examine particular results made this year in the formal representation of time, including time "intervals" and time "points", the first formalization of the plan recognition process, the definition of two distinct types of abstraction and an investigation of their properties and a new representation of action that allows context-dependent effects. In addition, we have completed the first stage of the RHET system, a practical knowledge representation system that not only has a formally well defined semantics, but also has a well-defined inferential capability.

7A.3 Time-Oriented Problem Solving

7A.3.1 Time Points and Moments

Interval-based temporal reasoning, since its introduction by the PI five years ago, has proven to be a useful formalism in many different areas. The purpose of this current project was to provide a concise formal theory for the interval logic and to investigate the relationship between interval and point based temporal representations. In conjunction with Pat Hayes, now at Xerox PARC, a new theory has been developed with the following features:

- The introduction of time periods as the basic building block, which does not commit to either being an interval or a (certain form of) point.
- A concise axiomatization (five axioms) that completely capture the inferential capability of interval-logic.
- The development of two distinct notions of point-like time periods
 - Points, which are endpoints of periods and can be shown not to be periods themselves but are useful for representing times of transition
 - Moments, which are non-decomposable periods, and are useful for representing "instantaneous" events such as flashes, clicks, etc.
- Demonstrating that our logic is consistent with various models of discrete time, continuous time, or models that mix the two.

Two of the papers in the appendix present this work in detail.

7A.3.2 A Formal Theory of Plan Recognition

Research in discourse analysis, story understanding, and user modeling for expert systems has shown great interest in plan recognition problems. In a plan recognition problem, one is given a fragmented description of actions performed by one or more agents, and expected to infer the overall plan or scenario which explains those actions. This project developed the first formal description of the plan recognition process.

Beginning with a reified logic of events, we designed a scheme for hierarchically structuring a library of event types. A semantic basis for non-deductive inference, called "minimum covering entailment", justifies the conclusions that one may draw from a set of observed actions. Minimum covering entailment is defined by delineating the class of models in which the library is complete and the set of unrelated observations is minimized. An equivalent proof theory forms a preliminary basis for mechanizing the theory. Equivalence theorems between the proof and model theories are presented. Minimum covering entailment is related to a formalism for non-monotonic inference known as "circumscription". Finally, the thesis describes a number of algorithms which correctly implement the theory, together with a discussion of their complexity.

The theory has been applied to a number of examples of plan recognition, in domains ranging from an operating system advisor to the theory of speech acts. The thesis shows how problems of medical diagnosis, a similar kind of non-deductive reasoning, can be cast in the framework, and an example previously solved by a medical expert system is worked out in detail. It has also been ported to BBN Labs for use in their FRESH/OSGP system (Naval Search and Rescue domain).

The analyses provides a firm theoretical foundation for much of what is loosely called "frame based inference", and directly accounts for problems of ambiguity, abstraction, and complex temporal interactions, which were ignored by previous work. The framework can be extended to handle difficult phenomena such as errors, and can also be restricted in order to improve its computational properties in specialized domains.

7A.3.3 Abstraction in Planning

The research conducted this year focused on issues involving abstraction in planning. In order for automated agents to be of use, there is a need for them to model, reason, and understand the world at a level sufficient to satisfy abstract goals within flexible, dynamic environments. It is clear to most researchers in artificial intelligence that this will require the use of expressive syntactic languages, e.g., first-order logic, in order to fully meet these knowledge requirements. As a consequence, however, the computational tasks associated with manipulating these languages is overwhelming. As such, there is an increasing awareness that agents will require the use of simplified, abstract models of the world to solve a great deal of their problems quickly, and that much of their expertise will be reflected by their choices of which simplified models to utilize. It is not always practical for human system designers to pre-specify which simplifications will maximize the agent's effectiveness. Therefore, it is important to define in a formal sense the exact nature of the abstraction process, so that eventually automated agents can perform their own abstractions. It is toward this end that this year's work was directed.

Abstraction typically involves removing details from a model in such a way that the remaining representation continues to have internal coherence, as well as a clear relationship to the domain that is being modeled. My work centered around defining two different types of abstraction within a symbolic planning representation, STRIPS. STRIPS is a representation that views the world as being in a particular *situation* at each moment. The actions of the agent are modeled by *operators* which can be applied to a situation satisfying its *preconditions* in order to have the desired *effects*.

Initially, work was performed in order to provide STRIPS with a model-theoretic semantics. Although this representation has been extensively used for over 1½ decades, it lacked a clear statement of either its syntax or semantics. A first-order semantics was provided by specifying a set of rules that translate a STRIPS system into an equivalent set of sentences in first-order predicate calculus. The equivalence is demonstrated by proving that the translation has a semantic model in exactly those cases in which the STRIPS system will never generate an inconsistent situation through the application of sequences of operators whose preconditions are satisfied.

The first type of abstraction involves simplifying the model by relaxing the conditions under which the operators can be applied. This work is an extension of

work by Sacerdoti at SRI in the early 1970's. Problems associated with this are that it is difficult to ensure that the world model still has coherence. In fact, the naive approach suffers from just these problems. My solution was to consider the world as being composed of causally partitioned subtheories, each of which bears on the operators. In relaxing constraints on the preconditions, one does so only along these partition boundaries. For instance, one can partition the set of sentences relating to the robot from those relating to the objects that it manipulates, and in this way suppress the position of the robot in searching for an abstract solution to object-moving problem. In this way, one is able to abstract in much the same way as in Sacerdoti's system, and in addition, one is guaranteed that the models one generates at abstract planning levels are never inconsistent. In addition, it is shown that these abstract systems have the *upward-solution property*, which states that if there exists a solution to the problem in the original problem space, there exists an abstract solution to an abstraction of the problem.

The second type of abstraction involves abstraction by analogy. In particular, the predicates used to describe objects and relationships can be mapped into abstract predicates to reflect similarity. So, for instance, *bottles* and *boxes* can be mapped to *containers*, with the axioms that appear in the abstract theory being only those that are common to the original predicates. Thus, one cannot conclude that containers are made of glass, since this is only true of bottles, although it can be concluded that all containers hold things. These analogical mappings are extended to operators, so that those operators performing similar actions on boxes and bottles (getting stuff in, removing stuff) can be considered an abstract action on containers. Planning can then proceed at the abstract level by attempting to first solve the problem by the abstract operators on the abstract world description. If a plan is found, each abstract operator is specialized by one of the operators that maps to it. As in the previous type of abstraction, particular care is taken in ensuring that consistency is preserved in the abstract spaces. In addition, it is shown that this type of abstraction has the *downward-solution property*, which states that if there exists an abstract solution to an abstraction of the problem, then there exists a solution to some specialization of the problem. There is also an intuitively appealing model-theoretic interpretation of the abstraction. For every semantic model M of the original problem space, there exists a model M' of the abstract space where the *extension* of each abstract predicate in M' is the union of the extensions of the original predicates in M . So, for instance, for any model of a theory containing bottles and boxes (and perhaps other containers), there exists a model of the abstract theory, where those objects which are containers are exactly those objects which are either bottles or boxes or in the extension of one of the other predicates from the original theory that maps to container.

Expectations in 1988

The planned research is to extend some of the issues raised in my dissertation on Abstraction in Planning. In particular, there is little formal analysis that has been performed that quantifies the performance gains that can be made in using abstraction spaces in search. The work that has been performed is either of a heuristic nature or is within weaker, state-space representations. This also points out the need to specify those strategies that exploit the presence of abstractions.

One of the focuses of my dissertation was in mapping first-order theories of a domain to simpler first-order theories. Another approach that I will be exploring is to map into a different representation *language* in order to exploit the better

performance associated with solving problems associated with the class of problems expressible in this language. For instance, suppose that one has a first-order representation of a robot's environment. It is to the agent's detriment to use a theorem-prover to perform path planning through a set of rooms connected by doorways if the agent has available to it a graph structure representing the connectivity of the rooms, since graph searches are almost trivially easy. The task, then, is to specify how an agent can recognize the presence of subparts of its domain model represented in an expressive language, such as first-order logic, that will map into a simple representation, such as graphs, upon which very fast algorithms are known to exist.

7A.3.3 Categories and Planning

Extensive psychological work has been done on the content and function of categories in our perception of the world. Recently, the emphasis has shifted from looking for categories implicit in the objects themselves to categories formed by the interaction between the agent and the objects. This has been loosely discussed as "goal-based" or "context-dependent" categorization. This project was based on the observation that an agent's appreciation of the relevance of object properties to its goals is what planning is about. We showed how functional categories in the form of lambda expressions can be extracted from the preconditions of valid plans to achieve the same goals. For example, although pens and typewriters have remote physical similarities, they both take part in plans that create a written note. More importantly, either may fulfill the same role of WRITING-INSTRUMENT in an abstract plan that creates a written note. This framework allows an agent to ignore superfluous similarities and focus on often subtle function similarities.

The notion of abstraction is important when considering how to limit explosive search. In an effort to understand more about the relationship between more or less abstract theories, we developed a model-theoretic view of an abstraction relation between first-order logical theories in terms of their model sets, or equivalently their theorem sets. We then looked at simple axiom deletion and disjunction introduction syntactic operations on theories and showed that they (non-strictly) weaken them, creating consistent abstracted theories. I then proved two key theorems demonstrating the power of these syntactic operations, including what assumptions must be made to generate any abstraction of a given theory.

The traditional STRIPS Assumption stipulates that action representations are self-contained. In particular, this disallows context-dependent effects and therefore does not address the Ramification Problem. This limitation also appears in some recent formal theories of action. We formulated a new formal theory that addresses the competence aspect of the Ramification Problem through a different representation of action based on basic actions that do not have a fixed context. With appropriate definitions and a single axiom, important laws of change and inertia become true that allow the determination of resulting states. In addition, the temporal model allows co-occurrence and non-occurrence of actions.

7A.4 Rhetorical

The Project

Rhetorical (Rhet) is a programming/knowledge representation system that offers a formally well defined set of tools for building an automated reasoning system. It's emphasis is on flexibility of representation, allowing the user to decide if the system will basically operate as a theorem prover, a frame-like system, or an associative network.

Rhet offers two major modes of inference: a horn clause theorem prover (backwards chaining mechanism), and a forward chaining mechanism. Both modes use a common representation of facts, namely horn clauses with universally quantified, potentially type restricted, variables, and use the unification algorithm. Additionally, they both share the following additional specialized reasoning capabilities:

1. variables may be typed with a fairly general type theory that allows a limited calculus of types including intersection and subtraction;
2. full reasoning about equality between ground terms;
3. reasoning within a context space, with access to axioms and terms in parent contexts;
4. the user is allowed to specify specialized reasoners for determining if two objects of a special type will unify;
5. escapes into LISP for use as necessary.

Rhet builds on some fairly recent work on PROLOG for efficiency, including such features as 'limited' intelligent backtracking.

Accomplished in 1987

By the end of the year, we had a subset of the full Rhet functionality operational. In particular, we currently support:

- Reasoning in Multiple Contexts¹.
- Unification between typed Rhet objects, as well as LISP objects.
- E-unification (equality) is fully supported contextually.
- A preliminary high level user interface (including a window interface) has been implemented.

¹Currently this is not completed at the user interface level

- Semantics for structured type reasoning has been defined (this is like frames).
- Both backward and forward chaining modes are operational.
- A number of useful builtin functions have been provided.
- The interface to the LISP interpreter/compiler has been completed.
- Constrained variables (posting proofs) is fully supported.

In addition we will release two TRs in January '88, one the user manual for the system, and the other lower level documentation for programmers who will maintain or modify the system.

Expectations for 1988

In 1988 we hope to also add:

- An operational structured type subsystem. This will include modifications as needed to the unifier to understand the implicit inverse relationship between, say, constructor functions and role functions. Implementation of this subsystem is currently in progress.
- Handle inequality reasoning. This will allow a user to declare two objects to be definitely not equal, so the reasoner will not attempt to solve a proof by assuming they are equal.
- Context operators at the user interface level (e.g. belief modal operators).
- Default slot values for our structured types.
- Goal Caching and Intelligent Cache Flushing, allowing a substantial performance improvement.
- Left Recursion detection.
- Axiom indexing (fast axiom selection). Mainly, a static approach, unlike incremental hashing.
- Axiom declarations (allows user to specify even better indexing by declaring variables to be locals, globals, downward, etc.).
- Editor interface (including incremental compilation). The current high level interface is incomplete, and we will want to expand it as we get more live users.
- An Axiom Compiler. This could give roughly anywhere from 1 to 2 orders of magnitude performance increase, if done properly.
- Handle limited set reasoning.
- Incremental hashing (a performance issue for KB lookup).

Publication List

Allen J.F. *Natural Language Understanding*. New York: Benjamin Cummings Publishing Co., Inc., 1987.

Allen, J.F. and P.J. Hayes, "A common-sense theory of time," TR 180, Computer Science Dept., U. Rochester, to appear, November 1987.

Allen, J.F., "Speech Acts," in S. Shapiro (Ed). *Encyclopedia of Artificial Intelligence*. New York: John Wiley and Sons, Inc., 1987.

Hartman, L., and J. Tenenberg, "Performance in practical problem solving," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, pp. 535-540, Milan, Italy, August 1987; TR 203, Computer Science Dept., U. Rochester, June 1987.

Hayes, P.J. and J.F. Allen, "Short time periods," *Proc., 10th Int'l. Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

Kautz, H., "A formal theory of plan recognition," Ph.D. Thesis and TR 215, Computer Science Dept., U. Rochester, May 1987.

Litman, D.J. and J.F. Allen, "A plan recognition model for subdialogues in conversations," *Cognitive Science* 11, 2, pp. 163-200, 1987.

Pelavin, R. and J.F. Allen, "A model of concurrent actions having temporal extent," *Proc., Nat'l. Conf. on Artificial Intelligence (AAAI)*, Seattle, WA, July 1987.

Tenenberg, J.D., "Preserving consistency across abstraction mappings," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, Milan, Italy, August 1987; TR 204, Computer Science Dept., U. Rochester, June 1987.

A Common-Sense Theory of Time

James F. Allen and Patrick J. Hayes
Departments of Computer Science and Philosophy
University of Rochester
Rochester, NY 14627

Abstract

The literature on the nature and representation of time is full of disputes and contradictory theories. This is surprising since the nature of time does not cause any worry for people in their everyday coping with the world. What this suggests is that there is some form of common sense knowledge about time that is rich enough to enable people to deal with the world, and which is universal enough to enable cooperation and communication between people. In this paper, we propose such a theory and defend it in two different ways. We axiomatize a theory of time in terms of intervals and the single relation MEET. We then show that this axiomatization subsumes Allen's interval-based theory. We then extend the theory by formally defining the beginnings and endings of intervals and show that these have the properties we normally would associate with points. We distinguish between these point-like objects and the concept of *moment* as hypothesized in discrete time models. Finally, we examine the theory in terms of each of several different models.

Introduction

The literature on the nature and representation of time is full of disputes and contradictory theories. This is surprising since the nature of time does not cause any worry for people in their everyday coping with the world. What this suggests is that there is some form of common sense knowledge about time that is rich enough to enable people to deal with the world, and which is universal enough to enable cooperation and communication between people. In this paper, we propose such a theory and defend it in two different ways.

First, the theory is powerful enough to include the distinction between "intervals" (i.e., times corresponding to events with duration), and "points" (i.e., times corresponding to instantaneous events), as well as allowing substantial reasoning about temporal ordering relations (including the abilities described in [Allen, 1984]). In addition, it includes a formalization of the beginning and ending of events by introducing the corresponding beginning and endings of times. We show that beginnings and endings act in many ways like "points," yet can be distinguished from them.

This research was supported in part by the Office of Naval Research under grant N00014-80-C-197, the National Science Foundation under grant DCR-8351665 and the US Army under grant DAAK21-84-C-0089.

Second, the theory has as allowable models a number of the temporal models that are suggested in the literature. This includes models that equate time with intervals and points on the real number line, models that hypothesize discrete time, and any model which mixes real points and intervals. Our claim is that if our common-sense theory of time excluded any one of these models, then there would be no debate as to whether that model was valid, since in that case our own primitive intuitions on the matter would be extremely clear. We do make one restriction on the models considered: they must allow the possibility that two intervals *MEET*, which is defined as the situation where there is no time between the two intervals, and no time that the intervals share. The importance of this relationship for naive theories of time has been argued elsewhere (e.g., [Allen, 1983; 1984]), and so will not be defended again here. Even with this requirement, we shall see that substantially different models are possible.

One important intuition which guides us is that time is occupied by events. If the universe did not change, there would be no time. Any sort of event or happening which can be described or thought of has a corresponding time, and the universe of times consists of these. We will often appeal to this intuition, which notoriously sometimes indicates continuity and sometimes discreteness. (In particular, it is the source of the need to allow time intervals to be able to MEET.)

In Section I, we axiomatize a theory of time in terms of intervals and the single relation MEETS. It is then shown in Section II that this axiomatization subsumes the interval-based theory proposed in [Allen, 1983; 1984].

We then extend the theory in Section III by formally defining the beginnings and endings of intervals and show that these have the properties we normally would associate with points. In Section IV, a distinction is made between these point-like objects and the concept of *moment* as hypothesized in discrete time models. Finally, in Section V, we examine the theory in terms of each of several different models.

This paper is a condensed version of a report, [Allen & Hayes, 1985], henceforth referred to as *the longer paper*, which presents additional discussion and the proofs for all the theorems below.

<u>Relation</u>	<u>Inverse</u>	<u>Definition</u>
BEFORE, b	AFTER, a	EXISTS k. I MEETS k MEETS J
=	=	EXISTS k,j. k MEETS I MEETS I & k MEETS J MEETS I
OVERLAPS, o	OVERLAPPED-BY, oi	EXISTS a,b,c,d,e. a MEETS I MEETS d MEETS c & a MEETS b MEETS J MEETS c & b MEETS c MEETS d
STARTS, s	STARTED-BY, si	EXISTS a,b,c. a MEETS I MEETS b MEETS c & a MEETS J MEETS c
FINISHES, f	FINISHED-BY, fi	EXISTS a,b,c. a MEETS b MEETS I MEETS c & a MEETS J MEETS c
DURING, d	CONTAINS, di	EXISTS a,b,c,d. a MEETS b MEETS I MEETS c MEETS d & a MEETS J MEETS d

Figure 1: The Relationships Between I and J in terms of the MEETS Relation

I. An Axiomatization of Interval Time

We start the formal development by positing a class of objects in our ontology that we shall call TIMES. These are intended to correspond to our intuitive notion of when some event occurs. We do not, at this early stage, make any commitment as to whether all times are decomposable or not.

The essential requirement of our intuition above is that two time intervals can MEET. We will take MEET as our primitive relationship between times and show that we can constructively define the complete set of possible relationships between intervals in terms of MEETS. Other reductions to a small set are possible; for example, Hamblin [1972] uses a relation we could define as less-than-or-MEETS.)

For example, we can define a relationship BEFORE to hold between intervals only if there exists an interval that spans some time between them. Thus

$$I \text{ BEFORE } J \iff \text{EXISTS } k. I \text{ MEETS } k \text{ \& } k \text{ MEETS } J.$$

As a notational convenience, we shall abbreviate conjunctions such as the above into a chain, i.e., $I \text{ MEETS } k \text{ MEETS } J$. We shall also use the abbreviations used in [Allen, 1983] for disjunctions between pairs of intervals. Thus "J (o oi s f d) I" is shorthand for the formula

$$(J \text{ OVERLAPS } I) \text{ OR } (J \text{ OVERLAPPED-BY } I) \\ \text{OR } (J \text{ STARTS } I) \text{ OR } (J \text{ FINISHES } I) \\ \text{OR } (J \text{ DURING } I).$$

All the possible relationships between times are defined in Figure 1. By including the inverses of these relations in the obvious way, we have thirteen relationships defined constructively in terms of MEET. Each entry

defines the ordered relation between I and J (I BEFORE J, I OVERLAPS J, etc.). The inverses are also between I and J and are equivalent to the original relationship between J and I (e.g., $I \text{ BEFORE } J \iff J \text{ AFTER } I$, etc.). The small letters listed with each give the abbreviation for the relation that will be used later in some examples.

With this reduction, we can axiomatize the interval logic entirely in terms of the MEETS relation, as follows.

The first two axioms are based on the intuition that intervals have a unique beginning position and a unique ending position. As a consequence of this, if two intervals both meet a third interval, then any interval that one meets, the other meets as well.

Axioms for Uniqueness of "Meeting Places":

- (M1) $\text{ALL } i,j. (\text{EXISTS } k. I \text{ MEETS } k \text{ \& } J \text{ MEETS } k) \implies (\text{ALL } l. I \text{ MEETS } l \iff J \text{ MEETS } l)$
- (M2) $\text{ALL } i,j. (\text{EXISTS } k. k \text{ MEETS } I \text{ \& } k \text{ MEETS } J) \implies (\text{ALL } l. I \text{ MEETS } l \iff J \text{ MEETS } l)$

The third axiom captures the notion of ordering. It simply states that given two "places" where two intervals meet, then these places are either equal or one precedes the other. This is axiomatized without referring to places as follows:

Ordering Axiom:

- (M3) $\text{ALL } i,j,k,l. (i \text{ MEETS } j \text{ \& } k \text{ MEETS } l) \implies$
- 1) $(i \text{ MEETS } l) \text{ XOR}$
 - 2) $(\text{EXISTS } m. i \text{ MEETS } m \text{ MEETS } l) \text{ XOR}$
 - 3) $(\text{EXISTS } n. k \text{ MEETS } n \text{ MEETS } j)$

In other words, we have exactly three possible cases, shown in Figure 2, for any four intervals i , j , k , and l .

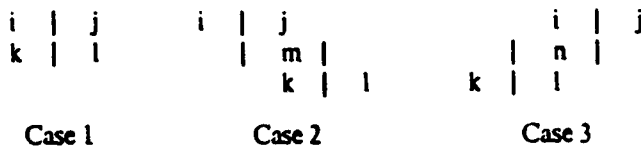


Figure 2: The Three Possible Orderings of i , j , k , and l in Axiom M3

Finally, we need some existence axioms. First, given any interval, there exists an interval that meets it, and an interval that it meets, i.e.,

- (M4) $\text{ALL } i \text{ EXISTS } j, k . j \text{ MEETS } i \text{ MEETS } k$
 i.e., $j \mid i \mid k$

A consequence of this axiom is that no infinite time intervals are allowed in our theory.

We need one more existence axiom, guaranteeing the existence of an interval which is the "union" or sum (+) of two adjacent intervals, defined by:

- (M5) $\text{ALL } I, J . I \text{ MEETS } J \Rightarrow$
 $\text{EXISTS } a, b . (I + J) .$
 $a \text{ MEETS } I \text{ MEETS } J \text{ MEETS } b$
 $\& a \text{ MEETS } (I + J) \text{ MEETS } b$
 i.e., $\begin{array}{ccccccc} | & a & | & I & | & J & | \\ & & & I+J & & & \end{array}$

Using the defined relations above, this axiom can be restated as

- $\text{ALL } I, J . I \text{ MEETS } J \Rightarrow \text{EXISTS } (I + J) \text{ such that}$
 $I \text{ STARTS } (I + J) \&$
 $J \text{ FINISHES } (I + J)$

We can prove that when $I + J$ exists it is unique, and that $+$ is associative.

With these five axioms and the definitions given in figure 1, the entire transitivity table for interval relationships given in [Allen, 1983] can be derived. Thus, this set of five axioms concisely captures that logic. This is not to say, however, that an implementation should not use the expanded set of relations. There are some important efficiency gains from the larger set of primitives, as described in [Allen, 1983].

II. Nests: Beginnings and Endings

There are classes of events described in English that cannot be associated with a temporal duration. These are often called "instantaneous" events, or "accomplishments" (e.g., [Mourelatos, 1978]). Thus, we can say "I closed the door," but if we say "I closed the door for three hours," it means we are repeatedly performing the action (contrast "I sat on the floor."). Similarly, a click, or the flash of a

strobe, cannot be qualified by a duration. Furthermore, the world after a click, or flash, could be essentially the same as before it, showing that these events cannot be identified with simple changes of state.

One common approach to handling the times for such events is to model them as points (real points in the *continuous* model; integers in the *discrete* model). In this section and the following one, we shall develop two distinct notions of points from our interval logic. These will be compared in the final section.

In this section we shall construct the equivalent of points within the interval logic defined in Section II by adopting a variant of filters, one of the standard mathematical constructions of points from intervals.

In particular, we define the beginning of an interval to be the set of all intervals that "touch the beginning" in any way, and the end similarly. We can define the ending of an interval similarly.

$$\text{BEGIN}(I) = \{ p \mid p \text{ (o s m f i d i e s i) } I \}$$

This can be defined solely in terms of the MEETS relation if one desires, but the above definition is simplest to understand. For convenience, we can define a *nest* as a beginning or an ending, and can now define relations over the set of nests which show them to have the properties of points. We shall say a nest N is *before* a nest M iff there is at least one interval in N that is before some interval in M .

for any two NESTS, N and M

$$N < M \Leftrightarrow \text{EXISTS } n, m . n \in N \& m \in M \& n < m$$

We show in the longer paper that nests have the important properties of points. The main result is that nests are totally ordered, i.e.,

Theorem 8: For any two nests N and M ,
 either $N < M$, $M < N$ or $N = M$

We can also show that the intuitive definitions of the interval relations in terms of nests are theorems. For example, we have

$$\begin{aligned} \text{BEGIN}(I) < \text{END}(I) \\ I \text{ MEETS } J \Leftrightarrow \text{END}(I) = \text{BEGIN}(J) \\ I \text{ OVERLAPS } J \Leftrightarrow \text{BEGIN}(I) < \text{BEGIN}(J) \& \\ \text{BEGIN}(J) < \text{END}(I) \& \\ \text{END}(I) < \text{END}(J) \end{aligned}$$

The second of these is especially important, as it shows that there is only one "place" where two meeting intervals actually meet. This is, perhaps surprisingly, a delicate matter. Very small changes in the definitions of BEGIN and END fail to achieve this. It is perilously easy to get a point structure, which distinguishes two "sides" of a single point, and other oddities, as discussed in [Van Benthem, 1982]. (We are grateful to Professor Dana Scott for bringing this and Hamblin's work to our attention, and emphasizing some of these subtleties.) We discuss this at greater length in the longer paper.

IV. Discrete Time and Time Points

We can now show that discrete time models introduce a different kind of "point" than the points that are defined above. In particular, discrete time hypothesizes times that are not decomposable. Let us introduce a distinction between *true-intervals* and *moments* as follows:

ALL I . TRUE-INTERVAL(I) $\Leftrightarrow$
 EXISTS a,b,c,d . a MEETS I MEETS d
 & a MEETS b MEETS c MEETS d

ALL I . MOMENT(I) $\Leftrightarrow$ $\sim$ TRUE-INTERVAL(I)

Thus, a true-interval has at least two sub-intervals (which might in turn be moments or true-intervals)--one that STARTS it and one that FINISHES.

Before we continue, it is important to remember that all of the earlier theorems were proven before any distinction was made between moments and true-intervals, so they all hold for both classes; none of the proofs ever depended on the decomposability of an interval. These definitions allow us to prove that two moments cannot overlap in any way, yet they can MEET each other. More precisely,

ALL I,J . MOMENT(I) & MOMENT(J) $\Rightarrow$
 I ($\prec$ m = m_i) J

Let us now consider the relationship between nests and moments. The definition of nests did not exclude nests defined at the beginning or ending of moments. In fact, we can show that the beginning of a moment is before the ending of that same moment! Thus, although a moment cannot be decomposed, we can distinguish its beginning from its ending.

We can also show that moments and nests cannot be considered to be isomorphic to each other. This is easily seen from the observation that moments can MEET each other, whereas nests cannot. Intuitively, a moment is a time *during which* some event (a flash, a bang) occurs, while a nest defines an abstract "position" in the sequence of times.

V. Discussion

It is interesting to interpret these axioms in various possible models. The simplest one is discrete time: intervals are pairs of integers $\langle n,m \rangle$ with $n < m$, and $\langle n,m \rangle$ MEETS $\langle m,k \rangle$. Then a moment is a nondecomposable interval $\langle n,n+1 \rangle$, and nests pick out integers, the places "between" moments. In this model there is a clear distinction between moments and points. We can also define several models based on the real line. For example, time intervals can be mapped into open or closed real intervals: however, then times can never MEET. A simpler continuous model, based on the integer model above,

defines time intervals as *pairs* of reals $\langle a,b \rangle$, with $\langle a,b \rangle$ MEETS $\langle b,c \rangle$. Following through the axiomatic definitions with this as a basis makes nests define points on the real line, as expected, but now there are no moments at all, since even the smallest interval is decomposable. We might try to extend the model to allow intervals of the form $\langle a,a \rangle$, which would qualify as moments, but now consider $\langle a,b \rangle$, $\langle b,b \rangle$ and $\langle b,c \rangle$. By our definitions, the first MEETS the last, yet they have the second between them, so the first is BEFORE the last, violating the ordering axiom. We have tried to fit real, substantial--though very small--time intervals into merely mathematical "places," and they don't fit.

However, another possible model is one which mixes these, using the same definitions of interval and MEET (from which all else follows) but allowing parts of the time line to be discrete and parts to be continuous. Intuitively, if we have only coarse time measuring tools available, then we can treat time as discrete, but the possibility always remains of turning up the temporal magnification arbitrarily far, if we have access to events which can make the finer distinctions, distinctions which can split "moments" into smaller and smaller parts.

Our axiomatic theory allows all of these models and others: it is uncommitted as to continuity or discreteness of the sequence of times, yet is powerful enough to support a great deal of the temporal reasoning of common sense.

References

- Allen, J.F., "Maintaining knowledge about temporal intervals," TR 86, Computer Science Dept., U. Rochester, January 1981; *Communications of the ACM* 26, 11, 832-843, November 1983.
- Allen, J.F., "Towards a general model of action and time," *Artificial Intelligence* 23, 2, July 1984.
- Allen, J.F. and P.J. Hayes, "A Common-Sense Theory of Time: The Longer Paper", TR Computer Science Dept, Univ. of Rochester 1985.
- Hamblin, C.L., "Instants and intervals," in J.T. Fraser, F.C. Haber, and G.H. Muller (Eds). *The Study of Time*. New York: Springer-Verlag, 1972.
- Mourelatos, A.P.D., "Events, processes, and states," *Linguistics and Philosophy* 2, 415-434, 1978.
- Van Benthem, J.F.A.K., *The logic of time*. Reidel, 1982.

Performance in Practical Problem Solving

Leo B. Hartman

Josh D. Tenenber

Computer Science Department
University of Rochester
Rochester NY 14627

Abstract

The quantity of resources that an agent expends in solving problems in a given domain is determined by the representations and search control strategies that it employs. The value of individual representations or strategies to the agent is determined by their contribution to the resource expenditure. We argue here that in order to choose the component representations and strategies appropriate for a particular problem domain it is necessary to measure their contribution to the resource expenditure on the actual problems the agent faces. This is as true for a system designer making such choices as it is for an autonomous mechanical agent. We present one way to measure this contribution and give an example in which the measure is used to improve problem solving performance.

1 Introduction

A primary goal of Artificial Intelligence research is to enable automated agents to have general reasoning abilities over a wide range of domains. Due to the inherent computational complexity of this task, system designers make *performance choices* that trade generality for improved resource use. Such performance choices are necessary for any agent with limited amounts of space and time to be effective within a dynamic world. Most running AI systems have embedded within them the choices that their designers felt would maximize their usefulness. Unfortunately, such choices rely on hidden assumptions, such as what variety and frequency of problems will be encountered, or how correct or close to optimal the eventual solution should be. Often, the designers themselves may not know what these assumptions are, since they are further obscured, for example, by choices implicit in the implementation of the representation language, or by insufficient prior information on the range of problems that the system may encounter. Unfortunately, this makes it difficult

for other researchers to determine if the tradeoffs are appropriate for their problem domain.

In this paper, we argue that performance choices should be made *explicit* in order to establish measures with which different choices can be compared. Further, we provide an example of how this might be done by developing a cost metric defined over search control strategies. The short-term benefit of this approach is that system designers will have a formalism which can be applied in determining preferred performance choices for the problems they are solving. The long-term benefit is that an automated agent will have the ability to evaluate its success in satisfying its goals in comparison with other candidate control strategies.

2 Intractability

Intelligent agents perform computations upon an internally stored *representation* of the world. Examples in [Levesque and Brachman, 1985] show how the choice of representation language bears on the computational complexity of solving problems using that language. As they point out, reasoning systems "will either be limited in what knowledge they can represent, or unlimited in the reasoning effort they might require." For example, if we use a first-order predicate representation language, then there exists no algorithm that will decide for any theory T and sentence S encoded in this language whether S is a theorem of T . However, it might be the case that a particular theory T in which we are interested can be encoded within a weaker representation, such as finite automata. Then there exist algorithms of bounded computational complexity that will answer most questions with regard to this theory, such as if a string is accepted by the automaton. Unfortunately, no algorithm exists that decides whether some first-order theory can also be expressed in a weaker representation language.

We will say that a problem P is intractable if there is no expression language L in which to state problem instances of P such that there is a deterministic Turing machine that can compute solutions to such instances within polynomial time and space. The following, a version of the travelling salesman problem, is an example of a problem believed to be intractable. Imagine that an agent finds itself in a city and is given a set of n buildings to visit, along with information as to the distance between each pair of buildings. The question to determine is whether the agent can visit each building exactly once on a path no more than k units long. It is possible that the agent can answer this question quickly for some given set of parameters, but

This work was supported in part by the NIH Public Health Service under grant 1 R01 NS 22407-01, by the National Science Foundation under grant DCR-8602958, and 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 Number F30602-85-C-0008 which supports the Northeast Artificial Intelligence Consortium (NAIC).

The authors would like to thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

it is believed that there is no deterministic algorithm (serial or parallel) that will solve every possible instance of this problem within an amount of time polynomial in the number of buildings in the problem. Note that this is not a function of the representation, but of the problem, since it is believed that there exists no representation for which this problem is tractable. Many problems that agents are called upon to solve are intractable.

The fact that an agent will encounter difficult problems does not mean that the agent should make no attempt to solve them. But in order to perform effectively, an agent must reason not about whether it can solve all possible problems optimally but, rather, how it can *maximize resource use over the entire sample of problems that it will encounter*. In order to do this, we believe that defining notions of *problem class* and *resource cost* is essential.

3 Problem Classes

Standard complexity theory typically defines a problem class in terms of the worst-case time or space behavior of a particular computational model, as a function of the size of the input problem. For example, the travelling salesman problem mentioned above falls in the class *NP-TIME*, which means that there is a non-deterministic Turing machine that solves every instance of a travelling salesman problem within time polynomial in the size of the input. The fastest known deterministic Turing machines for these problems take at least time exponential in the size of the input. Unfortunately, it is not known whether there are faster deterministic algorithms. It is in fact quite difficult to determine what is the fastest algorithm one can construct for a given problem.

There are several aspects of this definition of class which are inadequate for the task of optimizing an agent's use of its resources. First, each input problem instance to an algorithm is assumed to be in the correct form so that all operations of the algorithm are defined. For example, a typical sorting algorithm requires a list of elements from a set as input. The interpretation of this data structure is implicitly defined by the algorithm. Intelligent agents, however, are required to determine from general representations of its input which specific type of problem its input is an instance of. For example, if an agent has the goal of ordering a set of objects, it must recognize that this is a *sorting problem* and that there are algorithms for solving such problems. The difficulty of this recognition task will be a property of the language in which sorting problems are expressed.

Second, worst-case behavior is often too pessimistic. In practice the worst case may occur infrequently. Third, there might be restrictions one can place on the input that allow a subset of the problems to be solved quickly. For example, in the travelling salesman problem, if the cities are known to be collinear, then the solution can be trivially generated. Typically, general algorithms do not attempt to determine if the given input falls within one of the easier restricted subsets.

We will consider a much weaker notion of problem class. In general, an algorithm will define a problem class, in that a problem instance is a member of the

class if the algorithm solves that instance. We can assume that any complex agent will be called upon to solve a variety of tasks, some as easy as finding the largest element from a set, and some at least as difficult as the travelling salesman problem. If a domain is encoded using a general representation for which there exists an algorithm capable of solving each problem expressible in this representation, then we can consider an agent using this algorithm as solving problems from a single class. That is, every problem that the agent encounters falls within the large problem class defined by the algorithm.

Many algorithms fail to distinguish between tasks of various difficulties, as was noted earlier with the travelling salesman problem. Due to the general nature of the algorithms, a high computational cost is incurred when solving both the easy and the hard tasks. Taking a more general example, suppose an agent's domain is represented in the first order predicate calculus, and the agent is able to solve each problem it is presented, i.e., each theorem to be proven, using a complete proof procedure. Unfortunately, these proof procedures take greater than exponential time in the length of the input to generate a proof in the general case, if a proof exists. Assuming that some of the input problems involve finding the largest element from a set, the agent is expending considerably more resource than necessary to solve these problems using the general proof procedure. That is, if the agent were able to distinguish these problems at low cost, then the agent could use one of the known polynomial time algorithms for their solution. This fact motivates the use of, for example, *procedural attachment* in theorem proving [Nilsson, 1980].

An agent, then, can be provided with a repertoire of algorithms, each one optimized to a particular set of problems. The agent can be considered to encounter problems from a variety of classes - one problem class for each algorithm. Unlike the standard complexity-theoretic model where the algorithms are never required to distinguish between the various restricted subcases, we assume that complex agents will be obliged to do so. There will therefore be a cost associated with determining into which class a given input problem falls, i.e. which algorithm should be used to solve the problem. There might thus be no computational advantage to an agent in having a large repertoire of algorithms. The usefulness of a member algorithm will be a function both of the *frequency* with which problems falling within this algorithm's class are encountered by the agent, and of the comparative costs of solving problems using this algorithm versus other candidate algorithms. By considering the entire set of encountered problems as falling into a variety of problem classes, the statistical properties of the various classes can be exploited to improve performance. A faster running time of an algorithm whose class occurs with high relative frequency will improve the overall performance of the agent. The longer a period of time over which an agent solves problems, the greater will be the gains from the algorithm optimization, assuming the relative frequencies remain invariant. Hence such sustained problem solving activity justifies the expenditure of resource to perform the optimization.

4 Costs

There are few precedents in AI that involve measuring costs. One example is the A* algorithm [Nilsson, 1980]. A* is an algorithm for searching a *state space*, described by an initial state, a goal state, and the legal transitions between states, which computes a path between the initial and goal states. This algorithm has the characteristic that it is *admissible*, meaning that the first path it finds that solves the problem will be of minimum length, if any solution path exists. The algorithm works by searching through the current least-cost transition, where cost is a function of the number of transitions already taken on that path, and an estimate of the number of transitions required to reach the goal along that path. This cost, therefore, only measures the length of the solution. A* makes no attempt to model the resources required to arrive at a state, which includes the resources expended on the other incomplete or failed attempts up to that point in the computation. It is just such a measure of total computational costs that we are attempting to model. This replaces search for minimal *length* solutions with search for minimal *resource use* solutions.

A* places a premium on the optimality and correctness of every solution it reaches, which makes it generally *unsuited for the task of searching through complex domains*. It will typically be cost-effective for an agent to trade some degree of minimality, completeness and correctness of the solution for a speed-up in the time required to compute it. For example, several polynomial time algorithms have been constructed that find sub-optimal solutions to NP-complete problems [Garey and Johnson, 1979]. Many of these algorithms are such that the larger the upper bound on the acceptable distance between an approximate solution and the optimal solution, the faster the algorithm returns one of these approximate solutions.

Augmenting the symbolic approach of AI by decision theoretic techniques is suggested in [Feldman and Sproull, 1977]. It is argued there that for many problems in AI there are natural numeric measures of cost and that it is only on the basis of such measures that good decisions can be made. That is, a decision theoretic form is an appropriate representation for certain problems. In the search for a solution to a problem instance, the paper discusses how to select between alternate plans, how to deal with the uncertainty of the outcome of plan steps and how to introduce knowledge producing actions into plans. The basic mechanism is to compute the expected utility of a problem instance's candidate solutions. Our goal here is to investigate the expected utility, or more specifically, the expected computational expenditure, of a problem solving strategy over an agent's entire sample of problem instances. Similar to choosing, from a set of candidates, a solution that maximizes expected utility, we intend to choose from a set of candidate problem solving strategies one that maximizes expected computational cost.

Also discussed in [Feldman and Sproull 1977] is the utility of additional planning. If an agent possesses a solution to a problem then in order for further planning effort to be profitable, the improvements in the solution must offset the additional cost. In a similar way the

measurements and computation required to determine the expected computational cost of a strategy must be offset by the performance improvement that results from this improvement.

As mentioned earlier, finding lower bounds proofs is difficult for people, and will certainly be so for automated agents. This means that most agents will rarely be able to prove that they are doing the best possible. Therefore, it will usually be in the interests of an agent (and its designer) to express its performance choices in such a way that measuring the efficacy of these choices is possible. This requires considering all resource expenditure as cost, and expected improvements of one strategy over another as benefits. We claim that for agents who are called upon to make decisions in domains containing intractable problems, performance choices can only be made with respect to the sample of problem instances that an agent encounters. Whereas a particular performance choice may work quite well for a given problem sample, it may give equally poor performance for another sample. That is, agents cannot solve all problem instances optimally within such domains. By making the performance choices explicit, it is possible to evaluate whether a given choice is a good one for a particular sample. Thus, one is able to exploit any information about the sample of problems that is believed will be encountered in the future. Although the properties of future problems are unknown, predictions can be based upon the sample of problems that have already been seen. One of the long-term objectives of the approach advocated here is to state in a domain independent fashion those invariant properties of a particular domain or sample of problems that justify the belief that one has a good algorithm for the domain. Such domain independent properties also provide justification for applying the algorithm to other domains that are similar to the original domain with respect to these properties. Additionally, future performance improvements may also transfer across these similar domains.

5 Definitions

This section defines a cost metric for problem solving that illustrates some of the points raised above. This metric corresponds to the expected cost per problem that an agent expends. The metric is based on problem classes and the relative frequency that a given problem is a member of a particular class. The *solution strategy* used by an agent partitions problems into equivalence classes. One of these equivalence classes is intended to correspond to a set of problems of comparable difficulty or computational complexity. The cost metric makes clear how classes of difficult problems increase the expected cost and how an agent can exploit the presence of a class of easy problems.

We take the agent to consist of a search strategy. The agent exists in an environment in which problems are presented to it one at a time. The agent's response to the current problem is a solution to that problem obtained by the search strategy. We assume that the statistical properties of this sequence of problems presented to the agent are fixed. Specifically, we assume that the probability of members of a problem class being presented to the agent are constant and

that this probability is asymptotically approximated by the relative frequency of the problem class.

Let X be a countable set of problems. The problem instances that are presented to an agent are members of X . Let P be the set of solutions that correspond to the problems in X . We intend that P have a flexible interpretation. The members of P may be, for example, literal solutions as a sorted list is the solution to a sort problem. Alternatively the members of P may be algorithms that generate such literal solutions. However P is interpreted, the agent's problem-solving strategy selects members from P . Without loss of generality, every problem in X is assumed to have a solution in P . For the present purposes we also assume that whether a member of P solves a member of X is decidable.

Let s be the search function that the agent uses to generate solutions to problems. That is,

$$s : X \rightarrow P$$

Given a problem $x \in X$, $s(x)$ is a member of P that is a solution to x . Note that there may be several members of P that solve x but each search function selects only one such solution. Corresponding to each search function s is a function c_s such that the value $c_s(x)$ is the cost that s expends to solve problem x .

$$c_s : X \rightarrow R$$

The function c_s is real-valued but may be interpreted as the number of units of any arbitrary resource, e.g., time, space or food.

The strategy s partitions the problems into classes on the basis of the solutions chosen by s . We will consider two problems equivalent with respect to s if s returns the same solution for both. That is,

$$x \sim_s y \text{ iff } s(x) = s(y)$$

We use $[x]_s$ to denote the equivalence class of x under $\sim_s$.

Consider, for example, the situation in which X is a set of motion planning problems, the set of solutions P is a set of specialized motion planning algorithms each of which generates a sequence of robot actions. When the agent is presented with problem x , namely, a description of the environment and the robot's initial and final state, s selects some $p \in P$, an algorithm that yields a motion plan. The equivalence class of problem x , $[x]_s$, is just the set of problems for which s selects algorithm $p = s(x)$ to generate the motion plan. Without placing further conditions on P , the set of problems for which p generates correct motion plans will in general properly contain $[x]_s$, the set of problems for which s selects p as a solution. In general the actual cost of executing p is not included in $c_s(x)$ but is part of the cost of obtaining a literal solution for x . This point is discussed further in §6.

A problem sample is the denumerable sequence of problem instances presented to the agent. We represent a problem sample as a set of pairs $\langle i, x \rangle$ consisting of an index i and a problem instance x . If $\langle i, x \rangle \in H$, problem instance x is the i th element of the sequence. Thus, a problem sample H is a subset of $N \times X$ with the following properties:

$$\text{if } \langle i, x \rangle, \langle i, y \rangle \in H \text{ then } x = y$$

if $i \in N$ then there is some x such that $x \in X$ and $\langle i, x \rangle \in H$.

A prefix H^n of a problem sample H is just the set of problem instances up to some point n in the sequence:

$$H^n = \{ \langle i, x \rangle \mid \langle i, x \rangle \in H \text{ \& } i \leq n \}$$

We can extend the notion of problem equivalence classes to sets of problem instances and thus to problem samples and sample prefixes. For problem x and set of problem instances H

$$[xH]_s = \{ \langle j, y \rangle \mid \langle j, y \rangle \in H \text{ \& } y \in [x]_s \}$$

If H is a problem sample then $[xH]_s$ is the subset of the sequence whose members are in the equivalence class of x as determined by strategy s . That is, $[xH]_s$ is the set of problem instances in H for which s returns the same solution as it returns for x . If a particular problem x appears more than once in H , each pair that includes x is in $[xH]_s$. Similarly $[xH^n]_s$ is the set of instances in H^n of problems equivalent to x . We use $[H]_s$ to denote the set of all equivalence classes of problem instances of H with respect to s :

$$[H]_s = \{ [xH]_s \mid \exists i \langle i, x \rangle \in H \}$$

Similarly $[H^n]_s$ is the set of all equivalence classes of problem instances of H^n with respect to s .

Our cost evaluation of a strategy s for generating solutions is based on the above definition of equivalence classes of problem instances. The cost function for s can be extended in a simple way to members of a problem sample:

$$c_s(\langle i, x \rangle) = c_s(x).$$

Note that for problems x, y in the same equivalence class, we will not in general have $c_s(x) = c_s(y)$. Even though s obtains the same solution for x and y , s may expend more effort in one case than in another. Thus, for a prefix H^n the average cost of solving a problem instance in $[xH^n]_s$, denoted by $c_s[xH^n]_s$, is the sum of the cost of each individual problem divided by the size of $[xH^n]_s$. That is,

$$c_s[xH^n]_s = \frac{1}{|[xH^n]_s|} \sum_{\langle i, y \rangle \in [xH^n]_s} c_s(\langle i, y \rangle)$$

where $|A|$ denotes the cardinality of set A .

Let $\%(A|B)$ denote the relative frequency of members of set A in set B . For finite sets A and B , where A is a subset of B ,

$$\%(A|B) = \frac{|A|}{|B|}$$

Then the relative frequency with which problems in $[xH^n]_s$ appear in H^n is

$$\%([xH^n]_s | H^n) = \frac{|[xH^n]_s|}{|H^n|} = \frac{|[xH^n]_s|}{n}$$

Consider a single problem class $q \in [H^n]_s$. That is, let q be a set of equivalent problem instances in H^n with respect to search strategy s . The part of the average cost incurred by s on H^n that is attributable to q is just the product of the average cost that s expends on problems in q and the relative frequency of q in H^n :

$$c_s(q) \% \alpha H^n$$

Now summing over all such equivalence classes of problem instances, the average cost incurred by s to solve a problem in H^n is

$$C_s(H^n) = \sum_{q \in H^n} c_s(q) \% (q H^n)$$

If $c_s(q)$ is bounded for all problem classes q in H then from the assumption that the statistical properties of H are fixed we have that

$$C_s(H) = \lim_{n \rightarrow \infty} C_s(H^n)$$

is well defined and finite. This limit is the expected cost that s expends to solve a problem in H . For a particular s by recording the average cost and relative frequency of problem equivalence classes we can evaluate $C_s(H^n)$ and approximate $C_s(H)$. Thus we have some information about what performance we can expect from s on problem sample H . It is here that the invariance of problem class frequencies play a role. As shown in the next section this performance measure can be the basis for comparing and evaluating two different problem solution strategies.

6 Example: A hybrid algorithm

Having defined a measure of computational cost, the question arises as to how it may be used. The following example illustrates the use of this measure to improve performance. A problem solution strategy is defined such that, given a problem, it selects an algorithm whose output is the solution to the problem. From a family of such strategies we show how to choose the optimal one according to the performance measure.

Consider a sequence of algorithms $A_1 A_2 \dots A_n$, each member of which terminates on all input, takes as input an encoding of a problem and yields as output a solution to the problem or an indication of failure. Let the sequence have the additional property that any member of the sequence solves at least all of the problems of its predecessor. We refer to this property of a sequence of algorithms as the *subsumption property*. A specific instance of such a sequence of algorithms can be constructed to address the motion planning problem in robotics. The motion planning problem, e.g., [Lozano-Pérez, Wesley, 1979], is to find a collision free path for an object from an initial position to a desired final position. For an object B , if we have a sequence of object approximations $B_1 B_2 \dots B_n = B$ and a motion planning algorithm M , we can specialize M by each of the B_i 's to yield $M_1 M_2 \dots M_n$ so that M_i computes paths only for B_i . That is, M_i computes paths for the i th approximation of object B . If B_i strictly contains its successor in the sequence and if M is suitably well-behaved then the M_i 's have the subsumption property. That is, since M_i uses an approximation to the object that is bigger than that used by M_{i+1} , M_i can only find a path if M_{i+1} does (figure 6.1).

We are appealing to the intuition that giving up detail and *completeness*, i.e., the property that a solution is found if one exists, enables us to obtain more quickly the solutions we do find. In ascending order the B_i 's are increasingly better approximations to the original object. Similarly, the M_i 's are increasingly better

Sequence of object approximations



Situations distinguished by object approximations

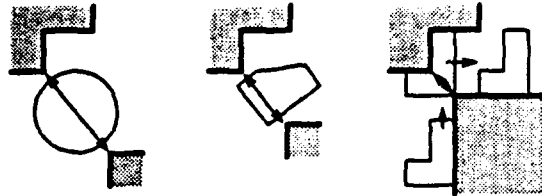


Figure 6.1 Object approximations generate motion planning algorithms

An example of a sequence of object approximations that can be used to define a sequence of motion planning algorithms with the subsumption property.

approximations to a complete algorithm to find motion plans for the original object. In this instance we may profit from giving up the precision of a complete algorithm by being able to use representations of lower combinatorial complexity.

Now let the search strategy s be such that, given a problem, s applies the A_i 's in turn until one of the A_i 's returns a literal solution to the problem. Such a search strategy, which we will refer to as a *hybrid algorithm*, is unambiguously specified by its component algorithms. Intuitively s has generality and also exploits the presence of simple problems. The strategy s can be as general as we want by appending to the sequence the most general known algorithm. In particular, s can be made complete if a complete algorithm is known. Because the simpler and, presumably, faster approximation algorithms are applied first, s solves easy problems quickly. To some degree a hybrid algorithm invests an appropriate amount of effort in solving a problem instance.

In the interests of applying the definitions of §5 we interpret s as selecting one of the A_i 's. That is, the solution that s returns is a member of $\{A_1 A_2 \dots A_n\}$. The A_i selected by s is one that can be executed to obtain a literal solution to the problem but trivially, by the time s selects A_i , we already have the literal solution that A_i generates. Now according to the previous definitions there is an equivalence class of problems associated with each of the A_i 's. Specifically, any problem x is in the equivalence class associated with A_i if A_i solves x (i.e., produces a literal solution for x), but no $A_j, j < i$, solves x . We refer to this equivalence class by $[A_i H]$, and note that $[A_i H]_s = \{x H\}_s$ for any problem x that s solves by selecting A_i .

As before $\%([A_i H^n]_s, H^n)$ is the relative frequency of members of $[A_i H^n]_s$ in H^n , the first n problems of sample H . If $c(A_i H^n)$ is the average cost of executing A_i on the problems of H^n then

$$c([A_i H^n]_s) = \sum_{j \leq i} c(A_j H^n)$$

This is the average cost that s incurs when s is applied

to H^n and selects A_i . The total cost $C_s(H^n)$ that s incurs on H^n is

$$C_s(H^n) = \sum_{a \in (A_1, \dots, A_n)} c_s(a|H^n) \cdot \mu(a|H^n, H^n)$$

Given this measure, it can be determined if each component algorithm is providing more benefit than the cost it is incurring. That is, although the expected cost of solving a given problem with any A_k , $k < i$, is less than with A_i , the cost of executing each such A_k is incurred in solving every problem from the class $\{A_i|H^n\}_s$. Classes that are too expensive or of low relative frequency can be dropped from s to yield a new strategy that has better overall performance. Given measurements of the relative frequencies and costs for the problem classes a dynamic programming method can be used to select the subset of component algorithms with lowest expected cost for the current problem sample [Hartman, 1987]. Note that if we take a hybrid algorithm that is optimized in this way for a particular problem sample and apply it to a sample with different statistical properties, not surprisingly, we are likely to obtain different performance.

In addition to obtaining a globally optimal set of component algorithms for a particular problem sample, we can also determine under what conditions an incremental change to s , e.g., the insertion of a new component algorithm into the sequence, yields an improvement on the expected cost per problem. That is, we know where candidates for future improvements may be found.

We do not claim that this is the best way to solve any particular kind of problem. The optimization is only within the narrowly defined class of hybrid algorithms. However, the example does show how the cost measure defined in the previous section can be used to optimize performance. In particular, the performance improvement is independent of the actual representations and component algorithms used. It does not depend on increasing knowledge about how to solve the particular problem such as, in this case, motion planning. The improvement is based solely on the invariant statistical properties of the problem sample and measurement of performance.

7 Conclusion

Agents in real world domains will be called upon to solve difficult problems. Although an agent might choose to expend arbitrarily large amounts of a resource in solving a particular problem, the opportunity costs of doing so typically make such a choice prohibitively expensive. This resource would be better spent in either solving approximations of this problem, ignoring this problem completely and solving other, easier problems, or improving the agent's general ability to solve these difficult problems. This is as true for agents who introspect upon their own problem solving ability as it is for designers who attempt to ascertain the utility of AI systems that they develop. We argue that these performance choices should be made explicit, and that they should be based upon the resource cost incurred in solving actual problem samples.

We have provided one example of a resource cost measure. Although by no means definitive, it measures the search costs associated with finding the solution to a given sample of problems using a given search strategy. As opposed to previous formal analyses of search strategies within artificial intelligence, the emphasis here is not upon the efficiency of the solution, as it is in A^* , but upon the amount of resources expended in finding solutions. Central to this endeavor is the concept of dividing a problem space into a set of problem classes. We can measure both the frequency with which a search strategy places problems from the sample within a class, and the average cost of solving each problem placed into a class. This allows one to demonstrate improved performance by changing the search strategy to place more of the problems into those classes for which solutions can quickly be generated, or to ascertain more quickly into which class a problem belongs. Cost measures such as the one described here, will be necessary, we believe, in order to maximize the resource use of an agent which must solve many problems drawn from an intractable class.

Acknowledgements

We would like to thank our advisor, Dana Ballard, for his patience and insight, and for the opportunity to do this research.

References

- Feldman, J.A., Sproull, R.F., Decision Theory and Artificial Intelligence II: The hungry monkey, *Cognitive Science* 1, pp.158-192 (1977).
- Garey, M.R., Johnson, D.S., *Computers and Intractability*, W.H. Freeman, New York (1979).
- Hartman, L., The practical solution of geometric problems, Technical Report 199, Computer Science, University of Rochester, in preparation (1987).
- Levesque, H.J., Brachman, R.J., A fundamental tradeoff in knowledge representation and reasoning, in Levesque, H.J., Brachman, R.J. (eds.), *Readings in Knowledge Representation*, Morgan Kaufmann, Palo Alto CA (1985).
- Lozano-Pérez, T. Wesley, M.A., An algorithm for planning collision-free paths among polyhedral obstacles, *CACM* 22, 10 (1979).
- Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga Publishing, Palo Alto CA (1980).

SHORT TIME PERIODS

Patrick J. Hayes
Schlumberger Palo Alto Research
3340 Hillview
Palo Alto, CA. 94304

James F. Allen
Computer Science
University of Rochester
Rochester, NY. 14627

ABSTRACT

Earlier papers (Allen and Hayes 1985, 1986) described a compact axiomatic theory which provided a formal basis for temporal reasoning. This theory makes a sharp distinction between time points and periods of time. We show here, by considering possible models of the theory, that it can be slightly extended to give a better fit to intuition. In particular, the extended theory is preserved under changes of temporal granularity.

TIMEPERIODS AND MEETING

The original period theory (Allen 1984, Allen & Hayes 1986) used periods rather than points as its temporal primitive. (The terminology used earlier was 'interval' rather than 'period'. We have changed to avoid confusion with the mathematical use of 'interval'.) The thirteen possible relationships between periods, including for example overlapping, inclusion, and before are defined in terms of MEETS, which we will write as in infix colon, with $p:q:r$ meaning $p:q$ and $q:r$. The basic axioms of the theory are then as follows:

- M1 forall p, q, r, s . ($p:q$ and $p:s$ and $r:q$)
implies $r:s$
- M2 forall p, q, r, s . ($p:q$ and $r:s$) implies
($p:s$
xor exists t . $p:t:s$
xor exists t . $r:t:q$)
- M3 forall p . exists q, r . $q:p:r$
- M4 forall p, q, r, s . ($p:q:s$ and $p:r:s$)
implies $q:r$
- M5 forall p, q . $p:q$ implies
exists r, s . $r:p:q:s$ and $r:(p+q):s$

These five axioms are all the assumptions which the theory makes. For a longer discussion of their implications and justification, see (Allen and Hayes 1986).

A key intuition is that propositions are true or false during periods, rather than at points of time. This overcomes the problem of the 'divided instant' (Van Benthem 1982). Consider switching on a light, so that the proposition "light on" is first false, then immediately true. If the timeperiods are thought of as sets of points, then only artificial constructions can avoid the dilemma of there being a point at which the light is neither on nor off (or, worse, both on and off).

However, timepoints can be defined within the

theory as the 'places' where periods meet. In (Allen & Hayes 1986), a set-theoretic construction is given of points from periods, but we can introduce points directly, and it can be shown that the theory obtained is identical. We will use variable names $u, v, w, \dots$ for points, $p, q, r, \dots$ for periods, and introduce two functions START and END from periods to points.

- P1 forall p, q . $p:q$ iff $\text{end}(p) = \text{start}(q)$
- P2 forall u exists p . $u = \text{end}(p)$

It is possible now to show that points, whatever they are, have all the properties one would directly expect, such as being a totally ordered infinite set with no limits. We could also start with points as the primitive idea and define periods as pairs of points from the totally ordered set. In (Allen and Hayes 1986) we show how the natural construction of periods from points by forming pairs $\langle u, v \rangle$ is a proper inverse to the derivation of points from periods defined by these axioms, so that such intuitively compelling results as $p = \langle \text{start}(p), \text{end}(p) \rangle$ are provable from the axioms. These points, however, are mere mathematical abstractions, and not times at which an event occurs or when some proposition is true.

POINTS AND MOMENTS

Points in time are places where periods meet, but they aren't themselves periods, not even very short ones. A timeperiod is the sort of thing that an event might occupy: it has substance, while a point is merely an abstraction. Consider for example a ball tossed into the air on the one hand, and a flash of lightning on the other. The time of the ball's flight can be divided into two periods, one of rising, the other of falling. The ball spends no actual time at the top of its flight: it does not hover there for a period, no matter how small. The periods of rising and falling MEET each other directly. The point at their meeting is a conceptual abstraction, not a real physical timeperiod. In contrast, the time taken by a flash of lightning, although pointlike in many ways, must be a period because it contains a real physical event. Other things can happen at the same time as the flash, such as a photograph being exposed or a horse dying. This is how we characterize such very brief moments of time:

- forall p . Moment(p) iff not exists q, r . $p = q+r$
- ie, a moment is indivisible into subperiods.

Moments have many of the properties of points. For example, if a period has moments at its ends then they are unique, and they uniquely define the period between them, by M5. But they also differ in many ways. For example, being periods, they have distinct endpoints.

The theory developed so far is somewhat unintuitive on this matter of truth at points and the qualities of moments. On one hand, some points seem to be natural receptacles of truth, and on the other, it is sometimes natural to treat a moment as being more pointlike. For example, the tossed ball's vertical velocity is positive on the way up, negative on the way down, so must be zero somewhere, by the intermediate value theorem: and the point where these periods meet is the obvious candidate. Qualitative reasoning uses such points liberally, and it is hard to see how it could do without them. On the other hand, it seems unnatural to distinguish the start from the end of a lightning flash, but we must if it is to occupy more than a point.

We might try to rescue the formalism by 'inserting' a moment whenever a proposition needs to be true at a point. The intuitive argument here would be that since the ball is motionless, there must be a period - perhaps an 'infinitely short' one, whatever that means - for it to be motionless in. But this leads to the dual problem: just as points are too small to be occupied by events, moments (as we have defined them) are too long to be suitably infinitesimal. Since a moment is a period, it has distinct endpoints, even though no points inside it. We can't identify these points without denying the existence of the moment itself, for to identify them is to claim that periods before and after the moment MEET, by P1. Thus, the time the ball comes to rest and the time it starts to fall would be separated by an interval during which the ball would be hovering, an unacceptable consequence.*

We seem to need to be able to treat moments as points, and vice versa, to capture intuition better.

MAPPING MOMENTS INTO POINTS

Suppose we introduce a mapping which identifies the endpoints of moments, so that moments are mapped into points. This is what Hobbs (1985) calls a change of granularity. The tolerance relation which defines it in this case is that two periods are indistinguishable if their endpoints are at most a moment apart. We can define this as follows:

```
forall u,v. u==v iff u=v
                or Moment(<u,v>)
                or Moment(<v,u>)

forall p,q. p==q iff start(p)--start(q)
                and end(p)--end(q)
```

As Hobbs notes, the change of granularity is well-understood only in the case where the tolerance relation is transitive, and hence is an equivalence relation. It is easy to see that -- is transitive just when moments never meet:

M6 forall m,n. Moment(m) and Moment(n) implies not m:n

(for if two moments meet, then the start of the first does not -- the end of the second). If we add this to our axioms, then the extended theory is preserved under the blurring which -- introduces. We prove this by defining relations which 'imitate' those of the basic theory but with a moments 'blur' at their ends.

* This is sometimes what naive intuition predicts, in fact: the 'road-runner effect'.

forall p,q. p::q iff end(p)--start(q)

or equivalently

```
forall p,q. p::q iff p::q
                or exists m. Moment(m) and
                        p:m::q
                        or m=overlap(p,q)

forall p,q,m. p::q implies p++q = p+q
                and
                (Moment(m) and p:m::q)
                implies p++q = p+m+q
```

Now, let M1' through M6' be the result of rewriting M1 through M6 with ::, + and = replaced by ::, ++ and ==. Then we have

THEOREM. M1-M6 entails M1'-M6'

The proof is elementary but tedious and omitted here.

It is easy to see that if == is equality, then the relation :: reduces to the primitive : and similarly ++ becomes the same as +. Hence, the axioms M1'-M6' may be regarded as the theory of ==-equivalence classes.

This result shows that the theory with M6 applies uniformly at any level of coarseness in which moments are identified with their endpoints. We can reason about periods and move flexibly from one level of acuity to another, needing only a truth-maintenance system to keep track of the various granularity assumptions being made from time to time.

This seems to provide a solution to the intuitive problems with which we began. A moment can be treated as a point, or vice versa, without changing the axioms of the temporal theory. The extended theory is stable under changes of scale. Notice that since the blurred relation :: includes the basic meets relation, we could use both relations in a description, and the theory would still be preserved under :: collapsing, so that we do not have to treat all the moments as points or vice versa. Thus, in the balltoss example, the topmost point can be talked of as though it were a moment, but all other points retain their inferior status as real points which do not expand into moments and in which nothing can be asserted. Or as an alternative, we could claim that all points have this expandable status, and then be forced into describing the door's status at the moment it closes.

Intuitive arguments for the reasonableness of M6 include the idea of perceptual acuity. When two dots are close enough, we perceive them as one. Imagine three dots x,y,z with x and y so close that they would seem dot-like if seen alone, and similarly y and z. One now perceives a short line, a dash. Two adjacent spatial 'moments' become a short, dense, 'period'. Adjacent 'moments' cannot be perceived. Similarly, for example, a succession of flashes with no periods between them are seen as a continuous light filling an period, which is why the movies work.

However, it is not yet clear what the models of the theory with M6 are. In the next section we will show that they are similar to the usual model of continuous time as the rational or real line.

MODELS OF THE THEORY

The axiom M6 seems rather strange at first sight, since in the two most obvious categories of model of the basic theory, it is either vacuous or false. These are respectively the dense and discrete models. In the latter, time is clock time, a sequence of discrete ticks. While many nonstandard models exist, the simplest is of course the integers. In these models, every moment meets the next moment, and so M6 is as false as it can be, everywhere. On the other hand, in a dense model, every period can be split into consecutive subperiods, so no moments exist, and M6 is true but boring. However, there is a nontrivial class of models which are neatly characterised by M6.

In a dense model, the points must form an unbounded dense linear order. If the model is countable (and one always is) then this is the rational line, but everything here applies equally well to the reals. Periods then can be interpreted as open intervals, with $(a,b):(b,c)$ defining the meets relation. The fact that the intervals, considered as sets of points, don't contain their endpoints, is not relevant to the way in which they stand for periods in this model. An exactly isomorphic model is obtained by interpreting periods as closed intervals with $[a,b]:[b,c]$, for example.

Notice however what happens if we include ALL rational intervals, open and closed, and try to define MEETING in the intuitively straightforward way which takes account of intervals as sets of points. Take for example the intervals

(a,b) (a,b) (b,c) (b,c)

called, say, p,q,r and s respectively. Then we would want it to be true that $p:r$ and $q:s$, but not that $p:s$. It follows then (from M2 and a little work) that there must be a period t such that $p:t:s$. The only candidate for such an period is the singleton $\{b\}$, ie the closed interval $[b]$. Now this must be a moment, since evidently $[b]$ has no subperiods. So this interpretation is not a model of the dense theory. On the other hand, it certainly is not isomorphic to the integers of the discrete model. In fact, this is a (rather extreme) example of a third class of interpretations, in which moments are intuitively point-like.

The general form of a model of this sort - call it a PACKED model - over the rationals, is as follows. The domain of periods contains open rational intervals and perhaps some singleton closed intervals, and all intervals which can be generated by taking the interiors of the unions of the closures, and the simple concatenation, of other intervals. Thus if it contains (a,b) and (b,c) and $[d]$, then it will also contain (a,c) , (b,d) and (a,d) , for example. MEETS is defined as follows: if $[b]$ is not in the domain, then $(a,b):(b,c)$; but otherwise, meets is interpreted set-theoretically, so that $(a,b):[b]:(b,c)$ and $(a,b):(b,c)$, etc.. At one extreme, a packed model is simply a dense model when no closed singleton instants are present. At the other, it is the fully set-theoretical interpretation in which periods are rational intervals. But many others are possible, for example one in which the only singleton moments are the integers, or where parts of the line are dense, parts dense with moments; or where just a few isolated points are distinguished as being the interiors of moments.

These singleton intervals, being periods, have distinct endpoints. Thus in a crowded model, the axioms P1, P2 assert the existence of 'points' on either side of a rational point: $\text{start}(\{a\})$ and $\text{end}(\{a\})$. The granularity-coarsening mapping defined earlier can be seen as making the claim that this distinction is irrelevant.

It follows from the earlier result that there is no way of formalising the distinction between these point-intervals and more substantial moments within the vocabulary of the time theory so far developed. One can see this intuitively by observing that any model which contains such instants can be replaced by a model in which instants are real intervals, simply by 'stretching' the line apart at those positions and inserting a gap. Exactly the same sentences in the language would be made true by such a stretched model, since these enlarged gaps would contain no meeting-points referred to by the sentences of the theory, so would be moments just as before; and the way they interact with the other periods will be unchanged. In order to express the distinction, we would need to introduce some way of talking about the size or duration of periods, and then point-intervals are moments with no duration.

The overall picture is this. The axiom M6 forces us to interpret moments as things with no extent, merely pointlike objects, and also exactly permits us to systematically treat them as points within the same theory, by a simple change of granularity. Thus there can be a moment during which the tossed ball's velocity is zero, and we can consistently treat it as though it were a point. And when we examine the models of our assumptions, there turns out to be a single point, in fact, where the moment should be. And yet other periods can meet directly, when time is dense, without there being moments between them. So we can have our moments and eat them.

ACKNOWLEDGEMENTS

We would like to thank Peter Ladkin for many instructive discussions. Much of this paper is based on insights found in (Van Benthem 1982).

REFERENCES

- Allen, J.F. "toward a general theory of action and time". Artificial Intelligence 23,2. July 1984
- Allen, J.F & Hayes, P.J. "A common-sense theory of time" Proc. 9th IJCAI, 1985
- Allen, J.F & Hayes, P.J. "A common-sense theory of time", TR, University of Rochester, 1986
- Hobbs, J. "Granularity" Proc. 9th IJCAI, 1985
- Van Benthem, J. F. A. K. "The logic of time" Reidel, 1982.

A Formal Theory of Plan Recognition

Henry A. Kautz*
Department of Computer Science
University of Rochester, Rochester, NY 14627

TR 215
May 1987

This report reproduces a thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The work was supervised by Dr. James F. Allen.

* Current address:

AT&T Bell Laboratories
Murray Hill, N.J. 07974

Curriculum Vita

Henry Alexander Kautz was born in 1956 in Youngstown, Ohio. After obtaining the highest score statewide on the 1974 New York State Regents Scholarship Examination, he entered the Case Institute of Technology in 1974, and transferred to Cornell University a year later. He received both an B.A. in English and one in mathematics from Cornell in 1978, graduating with highest honors. Mr. Kautz worked as a systems analyst for a year before winning a fellowship to the creative writing program at the Johns Hopkins University in 1979. During this time he wrote two professionally produced plays, and was awarded a M.A. by the Writing Seminars in 1980.

That fall Mr. Kautz returned to computer science, enrolling at the University of Toronto in 1980, supported by the Connaught Fellowship for foreign students. He produced his Master's thesis, *A First-Order Dynamic Logic for Planning*, under the supervision of Professor Ray Perrault, and received an M.S. in Computer Science in 1982. The National Science Foundation selected Mr. Kautz for a three-year fellowship that year, and he returned to the United States, entering the Department of Computer Science at the University of Rochester. Mr. Kautz was a teaching assistant for professors Patrick Hayes and James Allen in the Fall of 1983 and Spring of 1984 respectively, and was a research assistant for James Allen, his thesis advisor, for 1982-1983, and 1984-1987. He held research appointments in the summer of 1983 at BBN Labs in Cambridge, and in the summer of 1984 at SRI International, in Palo Alto, California. Mr. Kautz published papers on a number of topics in Artificial Intelligence during his tenure as a graduate student, including three presented at the 1986 annual convention of the American Association for Artificial Intelligence. Mr. Kautz finished his PhD thesis while employed as a Knowledge Representation Consultant at Bell Laboratories in Murray Hill, New Jersey, in the Spring of 1987.

Acknowledgments

The National Science Foundation helped supported my graduate study under Fellowship Grant No. RCD-8450125 and Research Grant No. DCR-8502481. This work was also supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441, and the Air Force Office of Scientific Research, Bolling Air Force Base, Washington, D.C. 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC). I thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper. The final document was produced on an Apple Macintosh[®] and Laserwriter[®].

I thank my thesis advisor, James Allen, for the countless hours we spent in research meetings and seminars over the years. He was always quick to suggest new ideas, sketch solutions to problems I ran into, and to help separate the wheat from the chaff. The faculty and staff at the University of Rochester provided an exceptionally supportive and collegial environment for graduate students. Thanks for interesting classes and wide ranging discussions go to professors Patrick Hayes, Henry Kyburg, and Jerry Feldman.

Gideon Frieder helped me get my first enjoyable summer job, working for the University of Buffalo Computer Science Department, and was fundamental in my decision to go to graduate school in computer science. Alex Borgida first introduced me to work in A.I. at the University of Toronto, where I was fortunate enough to become the student of Ray Perrault, both of whom taught me to think clearly and rigorously. Mention must be made of Gordy McCall, who hosted the Canadian A.I. Conference in Saskatoon; afterwards I felt like an honorary Canadian citizen for life. I am extremely grateful to Ron Brachman and Bell Laboratories for putting me on the payroll while I finished this document.

Time spent with fellow students Leo Hartman, Diane Litman, Josh Tenenberg, Jay Weber, Paul Kates, and Robin Cohen provided food for thought, as well as many fine meals. Jim Mayer taught me everything I know about making pasta, for which I more than forgive his utter skepticism about "artificial intelligence".

My parents were a constant source of moral and financial support during the many years of graduate study. Thanks for always encouraging me to try for the best.

This thesis is dedicated (of course) to Christine, who helped me celebrate the highs and saw me through the lows of working on a PhD. Her constant support, understanding, encouragement, and love have made this not only possible, but truly worthwhile.

A Formal Theory of Plan Recognition

Henry Alexander Kautz

Abstract

Research in discourse analysis, story understanding, and user modeling for expert systems has shown great interest in plan recognition problems. In a plan recognition problem, one is given a fragmented description of actions performed by one or more agents, and expected to infer the overall plan or scenario which explains those actions. This thesis develops the first formal description of the plan recognition process.

Beginning with a reified logic of events, the thesis presents a scheme for hierarchically structuring a library of event types. A semantic basis for non-deductive inference, called "minimum covering entailment", justifies the conclusions that one may draw from a set of observed actions. Minimum covering entailment is defined by delineating the class of models in which the library is complete and the set of unrelated observations is minimized. An equivalent proof theory forms a preliminary basis for mechanizing the theory. Equivalence theorems between the proof and model theories are presented. Minimum covering entailment is related to a formalism for non-monotonic inference known as "circumscription". Finally, the thesis describes a number of algorithms which correctly implement the theory, together with a discussion of their complexity.

The theory is applied to a number of examples of plan recognition, in domains ranging from an operating system advisor to the theory of speech acts. The thesis shows how problems of medical diagnosis, a similar kind of non-deductive reasoning, can be cast in the framework, and an example previously solved by a medical expert system is worked out in detail.

The analyses provides a firm theoretical foundation for much of what is loosely called "frame based inference", and directly accounts for problems of ambiguity, abstraction, and complex temporal interactions, which were ignored by previous work. The framework can be extended to handle difficult phenomena such as errors, and can also be restricted in order to improve its computational properties in specialized domains.

Table of Contents

Curriculum Vita	7A-D-2
Acknowledgements.....	7A-D-3
Abstract.....	7A-D-4
List of Figures.....	7A-D-10
Poemes humoristiques sur l'AI.....	7A-D-11

Chapter 1

Introduction.....	7A-D-12
1.1 Motivation	7A-D-12
1.2 Overview of Thesis	7A-D-13
1.3 Related Work on Plan Recognition.....	7A-D-17
1.3.1 Story Understanding.....	7A-D-17
1.3.2 Discourse.....	7A-D-21
1.3.3 Intelligent Computer Environments.....	7A-D-25
1.4 Related Work on Medical Diagnosis.....	7A-D-27
1.4.1 INTERNIST and CADUCEUS.....	7A-D-28
1.4.2 A Set Covering Model	7A-D-30
1.5 Related Work on Non-Deductive Inference	7A-D-30
1.5.1 Probability Theory.....	7A-D-31
1.5.2 Default Logic	7A-D-34
1.5.3 Circumscription	7A-D-34

Chapter 2

Representing Event Hierarchies.....	7A-D-38
2.1 Language.....	7A-D-38
2.2 Representation of Time, Properties, and Events	7A-D-39
2.3 The Event Hierarchy.....	7A-D-41
2.4 Components of Event Tokens	7A-D-42
2.5 Acyclic Hierarchies and Compatible Types	7A-D-43
2.6 Example: The Cooking World	7A-D-43
2.6.1 Diagrammatic Form.....	7A-D-44
2.6.2 The Abstraction Hierarchy.....	7A-D-45
2.6.3 The Decomposition Hierarchy.....	7A-D-46
2.6.4 Describing Instances of Events.....	7A-D-47
2.7 Conditional Actions.....	7A-D-48
2.7.1 Representing Conditional Actions	7A-D-48
2.7.2 An Example.....	7A-D-50

Chapter 3	
Covering Models.....	7A-D-51
3.1 Model Minimization	7A-D-51
3.2 Completing the Abstraction Hierarchy	7A-D-52
3.2.1 Theorem 3.1 (Exhaustiveness).....	7A-D-53
3.2.2 Theorem 3.5 (Disjointedness)	7A-D-53
3.2.3 Theorem 3.7 (Unique Basic Types).....	7A-D-53
3.2.4 Theorem 3.9 (Abstraction Completeness)	7A-D-53
3.3 Completing the Decomposition Hierarchy	7A-D-54
3.3.1 Definition of Covering Model and C-Entailment	7A-D-54
3.3.2 Theorem 3.10 (No Useless Events)	7A-D-54
3.3.3 Theorem 3.11 (Component/Use)	7A-D-54
3.3.4 Theorem 3.13 (No Infinite Chains).....	7A-D-55
3.3.5 Theorem 3.14 (Decomposition Completeness)	7A-D-55
3.3.6 Theorem 3.15 (Computability of C-Entailment)	7A-D-55
3.3.7 Theorem 3.16 (Not Predicate Completion)	7A-D-55
3.4 Circumscription	7A-D-56
3.4.1 Theorem 3.17 (C-Entailment and Circumscription)	7A-D-56
3.5 Example: The Cooking World, Continued	7A-D-57
3.5.1 Exhaustiveness Assumptions (EXA).....	7A-D-57
3.5.2 Disjointedness Assumptions (DJA).....	7A-D-57
3.5.3 Component/Use Assumptions (CUA)	7A-D-58
3.5.4 A Simple Recognition Problem.....	7A-D-60

Chapter 4	
Minimum Covering Models.....	7A-D-62
4.1 Cardinality Minimization.....	7A-D-62
4.1.1 Theorem 4.1 (Minimum Cardinality Defaults).....	7A-D-63
4.2 Cardinality Circumscription.....	7A-D-64
4.2.1 Circumscription with Variables	7A-D-64
4.2.2 Theorem 4.2 (Cardinality Circumscription).....	7A-D-65
4.2.3 Example of Cardinality Circumscription.....	7A-D-65
4.3 Example: The Cooking World, Continued	7A-D-67

Chapter 5	
Incremental Recognition	7A-D-69
5.1 Deficiencies in the MC Model	7A-D-69
5.1.1 The Combinatorial Problem.....	7A-D-70
5.1.2 The Persistence Problem	7A-D-70
5.2 Refining the Model	7A-D-71
5.2.1 Incrementally Minimize Cardinality	7A-D-71

5.2.2 Sticky Covers.....	7A-D-72
5.2.3 Discourse Cues.....	7A-D-72
5.2.4 Likelihood Associates	7A-D-72
5.3 Model Theory for Incremental Minimization.....	7A-D-72
5.3.1 Minimum Covering Submodels.....	7A-D-73
5.3.2 Monotonic Incremental Recognition.....	7A-D-74

Chapter 6

Examples.....	7A-D-76
6.1 The Cooking World, Once More.....	7A-D-76
6.2 Indirect Speech Acts	7A-D-78
6.2.1 Representation	7A-D-78
6.2.2 Assumptions.....	7A-D-81
6.2.3 The Problem.....	7A-D-81
6.3 Operating Systems: Multiple Events	7A-D-84
6.3.1 Representation	7A-D-84
6.3.2 Assumptions.....	7A-D-86
6.3.3 The Problem.....	7A-D-87
6.4 Medical Diagnosis.....	7A-D-91
6.4.1 Representation	7A-D-91
6.4.2 Assumptions.....	7A-D-93
6.4.2 The Problem.....	7A-D-94

Chapter 7

Algorithms for Plan Recognition.....	7A-D-100
7.1 Directing Inference	7A-D-100
7.2 Explanation Graphs.....	7A-D-102
7.2.1 Basic Elements of an E-Graph	7A-D-102
7.2.2 Roles of Event Types and of Nodes	7A-D-103
7.2.3 Definition of an Explanation Graph.....	7A-D-104
7.3 Computing the Uses of an Event Type	7A-D-106
7.3.1 Use Abstraction and Specialization	7A-D-106
7.3.2 The Uses and Direct Component Relations.....	7A-D-107
7.3.3 Definitions of Uses	7A-D-107
7.3.4 Example.....	7A-D-108
7.4 Constraint Checking.....	7A-D-108
7.4.1 Equality Constraints	7A-D-110
7.4.2 Temporal Constraints.....	7A-D-110
7.4.3 Fact Constraints.....	7A-D-111
7.5 Overview of the Algorithms.....	7A-D-111
7.5.1 Explain.....	7A-D-111
7.5.2 Match.....	7A-D-112

7.5.3 Group.....	7A-D-113
7.6 Pseudo-Code.....	7A-D-113
7.6.1 Utility Functions.....	7A-D-113
7.6.2 Explain-Observation.....	7A-D-114
7.6.3 Match-Graphs	7A-D-117
7.6.4 Group Observations	7A-D-118
7.7 Description of Operation.....	7A-D-121
7.7.1 Explain.....	7A-D-121
7.7.2 Match.....	7A-D-123
7.7.3 Group.....	7A-D-126
7.8 Completeness and Correctness	7A-D-127
7.9 Complexity	7A-D-128
7.9.1 Explain.....	7A-D-128
7.9.2 Match.....	7A-D-129
7.9.3 Group.....	7A-D-129
7.10 Transcripts.....	7A-D-131
Chapter 8	
Conclusions.....	7A-D-132
8.1 The Three Levels	7A-D-132
8.2 Applicability.....	7A-D-133
8.3 Generability and Extensibility	7A-D-134
8.4 Two Unresolved Issues	7A-D-135
References.....	7A-D-137
Appendix A	
Proofs for Chapter 3.....	7A-D-143
Theorem 3.1 (Exhaustiveness).....	7A-D-143
Theorem 3.2	7A-D-144
Theorem 3.3	7A-D-145
Theorem 3.4	7A-D-146
Theorem 3.5 (Disjointedness)	7A-D-146
Theorem 3.6	7A-D-148
Theorem 3.7 (Unique Basic Types).....	7A-D-149
Theorem 3.8	7A-D-149
Theorem 3.9 (Abstraction Completeness)	7A-D-150
Theorem 3.10 (No Useless Events)	7A-D-151
Theorem 3.11 (Component/Use)	7A-D-152
Theorem 3.12	7A-D-153
Theorem 3.13 (No Infinite Chains).....	7A-D-154

Theorem 3.14 (Decomposition Completeness)	7A-D-154
Theorem 3.15 (Computability of C-Entailment)	7A-D-156
Theorem 3.16 (Not Predicate Completion)	7A-D-157
Theorem 3.17 (C-Entailment and Circumscription)	7A-D-158
Appendix B	
Proofs for Chapter 4	7A-D-159
Theorem 4.1 (Minimum Cardinality Defaults)	7A-D-159
Theorem 4.2 (Cardinality Circumscription)	7A-D-160
Appendix C	
Proofs for Chapter 5	7A-D-164
Theorem 5.1 (Non-Universal Conclusions)	7A-D-164
Theorem 5.2 (Incremental Recognition)	7A-D-165
Appendix D	
Temporal Constraint Logic	7A-D-166
Appendix E	
Transcripts	7A-D-169
Hunt/Rob Example	7A-D-169
Cooking Examples	7A-D-173
Operating System Examples	7A-D-181
Language Examples	7A-D-185
Appendix F	
Details of Correctness Proof of Explain	7A-D-193

List of Figures

1.1: Hunt/Rob Hierarchy	3
2.1: Cooking Hierarchy.....	34
6.1: Language Use Hierarchy.....	70
6.2: Operating System Hierarchy.....	76
6.3: Medical Hierarchy.....	83
6.4: Conclusions from Jaundice	86
6.5: Conclusions from Pallor	87
6.6: Conclusions from Jaundice and Pallor	87
7.1: E-Graph for MakeMarinara	92
7.2: Extended Cooking Hierarchy Detail	98
7.3: Multiple Inheritance Hierarchy	103
7.4: E-Graphs for MakeMarinara, MakeNoodles, and their Match.....	115
7.5: Combinatorially Explosive Hierarchy	120
8.1: Erroneous Events	125
E.1: E-Graph for Get-Gun	161
F.1: Relation of $E_c(nc)$ to $E_0(n_0)$	185
F.2: Case 1	185
F.3: Case 2	186
F.4: Case 3	187

Poèmes humoristiques sur l'AI

*If you're dull as a napkin, don't sigh;
Make your name as a "deep" sort of guy.
You just have to crib, see
Any old book by Kripke
And publish in AAAI.*

*A hacker who studied ontology
Was famed for his sense of frivolity.
When his program inferred
That Clyde ISA bird
He blamed – not his code – but zoology.*

*If your thesis is utterly vacuous
Use first-order predicate calculus.
With sufficient formality
The sheerist banality
Will be hailed by the critics: "Miraculous!"*

*If your thesis is quite indefensible
Reach for semantics intensional.
Your committee will stammer
Over Montague grammar
Not admitting it's incomprehensible.*

Chapter 1

Introduction

1.1. Motivation

Perhaps the central concern of Artificial Intelligence is to devise methods for representing and reasoning about actions and plans. While plan synthesis has received careful formal analyses [McCarthy & Hayes 69], the inverse problem of plan recognition (or action interpretation) has appeared in mainly empirical and domain-specific programs of research. These include work on story understanding [Bruce 81, Wilensky 83], psychological modelling [Schmidt 78], natural language pragmatics [Allen 83, Litman 84], intelligent computer system interfaces [Huff & Lesser 82], and strategic planning. In each case, one is given a fragmented, impoverished description of the actions performed by one or more agents, and expected to infer a rich, highly interrelated description. The new description fills out details of the setting, and relates the actions of the agents in the scenario to their goals and future actions. The result of the plan recognition process can be used to generate summaries of the situation, to help (or hinder) the agent(s), and to build up a context for use in disambiguating further observations. This thesis develops a formal analysis of plan recognition. The analysis provides a firm foundation for much of what is loosely called "frame based inference" [Minsky 75], and directly accounts for problems of ambiguity, abstraction, and complex temporal interactions, which were ignored by previous approaches.

Plan recognition problems can be classified as cases of *intended* or *keyhole recognition* [Cohen, Perrault, & Allen 81]. In the first case, but not the second, the observer can assume that the agent is deliberately structuring his activities in order to make his intentions clear. Recognition problems can also be classified as to whether the observer has complete knowledge of the domain, and whether the agent may try to perform erroneous plans [Pollack 86]. This thesis concentrates on keyhole recognition of correct plans, where the observer has complete knowledge. We will consider, however, some examples from discourse, which are cases of intended recognition.

An important preliminary step is to define the scope of inferences which must be treated by a theory of plan recognition. Plan synthesis can often be viewed as purely hypothetical reasoning (i.e., if I did A, then P would be true). Some attempts have been made [Charniak 85] to formalize plan recognition as a similar kind of hypothetical reasoning: infer a plan P, such that if the agent did P, then he would do the observed

action A. Only a space of *possible* inferences is outlined, and little or nothing is said about why one should infer one conclusion over another, or what one should conclude if the situation is truly ambiguous. (Such criticism also applies to work based on "plausible" inference [Allen 83, Cohen 84, Pollack 86].) We insist that a theory of plan recognition specify what conclusions are absolutely *justified* on the basis of the observations, our knowledge of actions, and other explicit assumptions. In fact, our framework allows one to draw conclusions based on the *class* of *simplest* plans which contain the observed actions. Finally, while planning can be formalized as pure deduction, to formalize plan recognition we must develop a non-deductive, non-monotonic¹ system of inference.

An advantage of this approach is that the model and proof theories apply to almost any situation. They handle disjunctive information, multiple concurrent (and unrelated) plans, steps "shared" between plans, abstract event descriptions, and both incremental and non-incremental recognition. The corresponding algorithmic theory may place restrictions on the form of information handled in order to gain processing efficiency. The search for more general and efficient algorithms can continue without the need to revise or reinvent the basic principles upon which the theory of recognition is founded.

The vocabulary that has been used to describe plan recognition varies considerably. We will speak uniformly of observations as *descriptions of events*. The observer's knowledge is represented by a set of first-order statements called an *event hierarchy*. The result of the recognition process is a description of the *end events*, those which are self-contained and self-justifying, which make up the situation. The wide applicability of the event vocabulary suggests that this work is relevant to areas of Artificial Intelligence not normally associated with planning or plan recognition, such as diagnostic reasoning. We will examine a simple medical-diagnosis problem using our logical machinery.

1.2. Overview of Thesis

An event hierarchy is a collection of restricted-form first-order axioms, used to define the abstraction, specialization, and functional relationships between various kinds of events. The functional, or "role/value", relationships include the relation of an event to its *component* subevents. There is a distinguished type-predicate, *End*, which

¹ A system is non-monotonic if some conclusions which may be drawn from a given set of assumptions may no longer be drawn from a larger set of assumptions.

holds of events which are not components of any other events. Recognition is the problem of classifying the End events which generate a set of observed events. The second half of this chapter reviews research in plan recognition in many different areas of Artificial Intelligence, as well as related work in non-monotonic inference.

Chapter 2 formally defines event hierarchies, and shows how they may be used to represent hierarchically-structured plans of action. It argues for the particular method of representation, and describes the expressive limitations of alternative systems. Event hierarchies can encode conditional actions without additional machinery. A hierarchy of actions involved in cooking is developed in detail, and serves as a basis for examples throughout the rest of the thesis.

An event hierarchy does not, however, by itself justify inferences from observations to End events. The lexical hierarchy justifies deductions from an event to its components; it does not rule out the possibility that those components may occur without reference to any of the End events mentioned in the hierarchy. Consider the following example. (The thick grey arrows denote abstraction or "isa", and the thin black arrows denote component or "has part".)

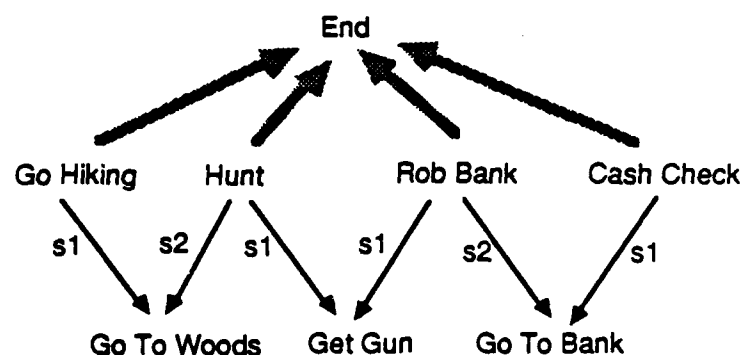


figure 1.1: Hunt/Rob Hierarchy

Suppose $\text{GetGun}(C)$ is observed. This statement, together with the hierarchy, H , does not entail $\exists x . \text{Hunt}(x)$, or $\exists x . \text{Hunt}(x) \vee \text{RobBank}(x)$, or even $\exists x . \text{End}(x)$. There are models of $\{\text{GetGun}(C)\} \cup H$ in which none of these statements hold. For instance (where we describe a model by listing its positive atoms), none hold in $\{\text{GetGun}(C)\}$, and only the last in $\{\text{GetGun}(C), \text{CashCheck}(D), \text{GoToBank}(s1(D)), \text{End}(D)\}$.

Yet it does seem reasonable to conclude that someone is either hunting or robbing a bank, on the basis of the given hierarchy. This conclusion is justified by as-

suming that the event hierarchy is *complete*: that is, whenever a non-End event occurs, it must be part of some other event, and the relationship from event to component appears in the hierarchy. This completeness assumption can be expressed by defining a special subclass of models of H, called "covering models". For the example, $\{\text{GetGun}(C), C=s1(D), \text{Hunt}(D), \text{GoToWoods}(s2(D)), \text{End}(D)\}$ and $\{\text{GetGun}(C), C=s1(D), \text{RobBank}(D), \text{GoToBank}(s2(D)), \text{End}(D)\}$ are covering models of H, but none of the other models described above are.

Chapter 3 uses the notion of a covering model is used to define a new semantic relation, called *c-entailment*. In this example, hunting or bank robbing occurs in all covering models in which an agent gets a gun, or formally:

$$\text{GetGun}(C) \vdash_H \vdash_c \exists x . \text{Hunt}(x) \vee \text{RobBank}(x)$$

The second part of that chapter relates *c-entailment* to ordinary entailment and deduction. An easily-computed "closure" function *cl* is defined, with the property that a statement is *c-entailed* by an observation if and only if that statement deductively follows from the observation and the closure of the event hierarchy. That is,

$$\Gamma \vdash_H \vdash_c \Omega \quad \text{if and only if} \quad \Gamma \cup \text{cl}(H) \vdash \Omega$$

Thus *cl*(H) axiomatically captures the class of covering models; or, equivalently, explicitly states the closure assumptions in effect when one uses an event hierarchy for recognition. The theory of *c-entailment* is related to John McCarthy's system of non-monotonic inference known as *circumscription*.

When several events are observed, still stronger assumptions are commonly employed. Suppose that $\{\text{GetGun}(C), \text{GoToBank}(D)\}$ is observed. This set does not *c-entail* an instance of robbery; the model containing an instance of hunting *and* an instance of check cashing provides a counterexample. By Occam's razor (do not multiply entities unnecessarily) we *would* be justified in concluding $\exists x . \text{RobBank}(x)$; this principle can be realized by distinguishing the *minimum covering models* of the observations. These models define a final semantic relation between observations and conclusions, *mc-entailment*. In the example,

$$\{\text{GetGun}(C), \text{GoToBank}(D)\} \vdash_H \vdash_{mc} \exists x . \text{RobBank}(x) \wedge C=s1(x) \wedge D=s2(x)$$

Chapter 4 develops the theory of minimum covering models and mc-entailment. Mc-entailment is shown to correspond to a particular case of *circumscription with variables*.

Chapter 5 extends the theory to describe *incremental* plan recognition. An incremental recognition process *cyclically* makes observations and infers consequences. The conclusions reached at the end of any particular cycle form the basis for those reached in the next cycle. Incremental recognition is *dichronic*: the conclusions reached may depend upon the order of the observations. While this may make it non-optimal for some recognition problems, we shall argue that it is a plausible description, on both computational and psychological grounds, of the kind of inference performed in plan recognition and its applications.

This formal framework shows how one can infer "up" an event hierarchy, and unify the explanations of several observations in order to reach a stronger conclusion. Very few restrictions are placed on the kinds of events which can be encoded: disjunctions may appear in plans or in observations, observations may be incomplete, and arbitrary temporal constraints may appear between events. Several examples of plan recognition are discussed in detail in Chapter 6, from the domains of cooking, natural language pragmatics, and operating system interfaces. The chapter concludes with a comparison of plan recognition to medical diagnosis. Diseases can be taken to be End events, and symptoms to be component events. A problem handled by a version of INTERNIST, a medical expert system, is recast in our framework.

Traditional work in artificial intelligence on high level² recognition problems has often relied on graph-matching algorithms. A lexical hierarchy is very naturally represented as a labeled digraph, and such graphs can be given a straightforward semantic interpretation. Chapter 7 describes some graph-based recognition algorithms which are justified by the notion of mc-entailment: that is, they compute structures which can be interpreted as statements true in all minimum covering models. The algorithms are useful and interesting in that they suggest what conclusions *should* be drawn, while the formal framework only specifies what conclusions *can* be drawn. There are three basic algorithms. The first constructs an AND/OR graph bottom up, from each observation to an instance of End. The second minimizes the set of End events, by performing a top-down match of the structures created in the previous step. The third algorithm controls the first two, and decides which observations are part of

²High level recognition problems involve the application of a great deal of specific world knowledge to a relatively small amount of symbolic (non-numeric) data. Low level recognition problems reverse this situation: there is a massive amount of quantitative data, interpreted by general principles about the physical nature of the domain.

the same of End event. Has the framework's power and generality been bought at the cost of computational intractability? We believe not. While it is easy to show that the *worst-case* cost of solving a plan recognition problem matches that of general deduction (namely, exponential on the size of the knowledge base), careful use of event *abstraction* significantly collapses the search space.

1.3. Related Work on Plan Recognition

Work in A.I. on plan recognition has concentrated on story understanding, discourse, and intelligent computer interfaces and environments. Each area is reviewed in roughly chronological order.

1.3.1. Story Understanding

1.3.1.1. Psychological Modeling

One of the first papers to explicitly invoke the phrase "plan recognition" was the report by [Schmidt, Sridharan, & Goodson 78] on the BELIEVER system. BELIEVER was designed to illustrate and test a psychological theory of "how descriptions of observed actions are utilized to attribute intentions, beliefs, and goals to the actor." Schmidt and his colleagues conducted experiments in which human subjects were presented simple descriptions of sequences of actions by a single agent. The sequences were interrupted at various points, and the subjects were asked to summarize the events so far, describe what the agent was trying to do, or predict what the agent would do next. The researchers observed that:

1. Summaries often included non-described, but expected, actions.
2. Subjects did not provide summaries which referred to a disjunctive set of plans, but they did provide "sketchy" summaries.
3. Subjects often provided summaries of the form: "The agent was *trying to do* (some act), but failed *because* (some reason)."

From these and similar observations, Schmidt concluded that people understood and remembered event sequences by recovering the implicit structure of causal relations between the events. Schmidt argued that plan recognition is a single-minded, hypothesis-driven process. Based on the initial observations (descriptions), the subjects

seemed to devise a single hypothesized plan (for the actor). This hypothetical plan would be incrementally revised and made more detailed as further observations were made.

BELIEVER implemented this strategy of plan recognition. Once the setting of the story was input, BELIEVER would retrieve a single parameterized plan from memory. As each observation was input, BELIEVER tried to match the description against an *expected* action (that is, an action in the plan whose preconditions were true). Failure of the observation to match an expected action would trigger "critics", that is, specialized pieces of code which revised the plan to account for errors and accidents by the agent.

The work on BELIEVER is quite different from our own. While BELIEVER was a "top down" inference system, we have concentrated on the "bottom up" aspects of plan recognition: if *many* different ultimate plans are possible, how do we home in on just a few possibilities? BELIEVER was offered as a rough psychological model, while we have concentrated on building a tight mathematical model of "ideal" performance. Schmidt's work began to deal with the critical issues of errors and accidents, which we have not examined. (Recent work by [Pollack 84] investigates problems in recognizing erroneous plans.) Both Schmidt and our work stress the use of a library of plans, and the importance of "sketchy" or abstract plans; our work extends BELIEVER's notion of abstraction, which involved plans with uninstantiated parameters, but not a hierarchy of abstract plan types.

After the BELIEVER project interest grew in devising systems that could understand stories which involved a number of different characters, each pursuing goals and plans which could interact and conflict with the plans of the other characters. [Bruce 81] presented a detailed analysis of the interacting plans in a number of children's stories, including *Hansel and Gretel*. Unfortunately Bruce did not continue this line of research with a computational model. The greatest nexus of interest in story understanding grew up in Roger Schanks' group at Yale.

1.3.1.2. Script Based Systems

Schank's theory of *scripts* was designed to account for all kinds of regularities in the world, both physical and social [Schank 75]. For example, the restaurant script tells us that restaurants typically have tables and chairs, and that a person in a restaurant typically has the goal of eating.

While scripts alone only provide a limited version of planning and plan recognition, [Wilensky 82] extended script theory to handle more dynamic domains. Wilensky argued that most everyday planning problems involve little or no search ("canned" plans are known for all possible goals) but do involve complicated interactions between goals. Furthermore, plan recognition is inextricably intertwined with plan synthesis. For example, given the following text:

*John wanted the newspaper. It was raining outside, so
John called for his dog Spot.*

Wilensky claimed one would conclude that John wants Spot to fetch the newspaper by reasoning as follows: Having the newspaper is one of John's goals. The standard plan for this is to go outside and get the newspaper, and so this standard plan is (tentatively) included in (the reader's beliefs about) John's wants. The fact that it is raining would invoke the *stay dry* goal, so this is added to John's wants. The reader then simulates John's planning: the goals conflict, so a *resolve conflict* goal is also created. There are many different ways that *resolve conflict* can be achieved, so the recognizer stops planning. Now the reader learns that John called for Spot. The effect of this is determined to be that Spot is with John. The reader tries to *connect* this new piece of input, as tightly as possible, with the current plan. It notices that one expansion of the *replan* meta-plan for the *resolve conflict* meta-goal is to alter the *get paper* plan so that some other agent goes outside and gets the paper. Making Spot the other agent connects this plan to the input.

While parts of this theory were implemented in many computer programs, much remained vague. A crucial feature were the so-called *text comprehension principles*, which were used to "tie together" the individual sentences of the story. These were *coherence*, meaning the character's plan is consistent; *least commitment*, meaning that a reader shouldn't prematurely assume that *any* particular explanation found is *the* explanation, and then have to undo it; and *parsimony*, meaning that a reader should "maximize the connections between the inputs". Specific versions of these general principles appear in our framework for plan recognition. Our model theoretic approach forces us to only consider consistent plans (inconsistent plans would have no model). Least commitment arises from concluding what holds in *all* minimum covering models, rather than a particular one. The principle of parsimony corresponds to our minimization of unrelated events.

1.3.1.3. Abduction

Recent work by [Charniak 85] views plan recognition, or "motivation analysis", as a kind of *abductive inference*. The term "abduction" comes from an early attempt in the philosophy of science to look at explanation as the process of finding the best hypothesis which logically entails the thing to be explained [Peirce 58].³ For example, suppose we know that all men are mortal:

$$\forall x . \text{man}(x) \supset \text{mortal}(x)$$

The thing to be explained is that Socrates is mortal. An abductive inference would take us from mortal(Socrates) to man(Socrates). Thus Socrates is mortal because he's a man. Abduction is obviously a *very* unsound rule of inference, that can lead to ridiculous conclusions. Pierce never claimed abduction was all that was involved in scientific explanation: it was merely a method of *generating* hypotheses that could be tested and weighed by other means.

Charniak's system employs abductive techniques to generate hypotheses which entail the (logical forms of) the sentences in simple stories. In order to limit the amount of inference, only a small subset of the reader's knowledge is assumed to be available at any one time. For example, a story might be:

John went to the store. He walked down the aisle and picked up the milk.

The individual words in the story, such as "store" and "milk", activate the semantically-related plan of *going grocery shopping* in the system's memory, as well as related axioms about how shopping carts and check-out lines work. The system then tries to generate a resolution-style *proof* that the story occurs, where it is allowed to make two kinds of assumptions:

1. An instance of any active concept may be assumed to exist. Thus the theorem prover may simply assume $\exists x . \text{Shopping-Trip}(x)$, if such an assumption is needed to close off a branch of the proof tree.
2. Any two terms may be assumed to be equal, if the system cannot prove (within some finite amount of time) that the terms are not equal. Thus the system may assume, for instance, that the go-to-store step of the Shopping-Trip event is equal to the particular instance of going to the store that is mentioned in the story.

³Eventually Peirce abandoned the notion of abduction [Hacking 83].

Charniak's framework is much less "cautious" than our own. If several different plans could entail the observations, it must choose a single one, whereas our framework would simply conclude the disjunction. It is also not entirely clear how well the technique of controlling inference by drawing activation from lexical items really works: it would be possible to "tune" such an activation network so that it activated just the formulas needed for any *particular* example. No semantic basis for the system is given, or indeed seems possible. Despite these criticisms, Charniak's work is among the most principled in the "scruffy" area of A.I.⁴

1.3.2. Discourse

1.3.2.1. Allen and Perrault

[Cohen 78] first formalized Austin's and Searle's speech act theory in terms of planning. Utterances were viewed as actions which transformed the beliefs of the speaker and hearer. [Allen & Perrault 80] extended the analysis to include plan recognition. Plan recognition is necessary to account for the fact that a speaker need not fully and literally execute a speech act in order to achieve its effect. Instead, the speaker need only perform an act which suggests to the hearer that the speaker's overall intention is to achieve the desired effect. Phenomena accounted for by plan recognition include indirect speech acts, understanding of sentence fragments, and certain kinds of context-dependent implicatures.

Allen analyzed plan recognition in terms of a set of plan recognition rules together with a heuristic control strategy. The rules isolate those inferences which are *plausible*, but not valid. The control strategy determines which of these inferences should actually be accepted. An example of these rules is *precondition/action* rule, which states that if (agent) H believes (agent) S wants proposition P to hold, then H may plausibly conclude that S wants action Act to be performed, given that P is a precondition⁵ for Act. Another is the *effect/action* rule, that states if S wants P to hold, and P is achieved by Act, then S *may* want Act to occur.

⁴ Workers in Artificial Intelligence are often divided into the neat and scruffy camps [Schank 83], with the neats trying to create formal theories [Kautz 86c] which systematize the heuristics uncovered by the intuition-driven scruffies.

precondition/action

H Bel S Wants P $\Rightarrow$ H Bel S Wants Act
given that P is a precondition of Act

effect/action

S Wants P $\Rightarrow$ S Wants Act
given that P is an effect of Act

Rules such as these were applied to form a chain of inference from a single observed action (called the *alternative*) to one of a number of possible contextually-dependent goals (called the *expectations*). Allen's system tried to find the "most likely" (or perhaps "most obvious") chain by performing a best-first search, numerically scoring chains of inference⁶ by rules like the following:

- (H1) Decrease the rating of a chain of inference if it contains an action whose preconditions are false at the time the action starts executing.
- (H3) Increase the rating of a chain of inference if it contains descriptions of objects and relations in its alternative that are unifiable with objects and relations in its expectation.

One of the most interesting rating heuristics is applicable only in *communicative* domains, where the observer can assume that the actor is *trying* to make the intentions behind his acts obvious. It says to *lower* the rating of a chain of inference if a great many different rules all apply to its last step. Thus ambiguity is penalized.

Allen did not try to provide any theoretical underpinning for his rules of plausible inference, in terms of probability or model theory. Nor did his system deal with multiple observations, which is the main focus of attention in our work (as well as that of [Litman 84], discussed in the next section). The basic notion of forming a chain connecting an observation to a goal seems related to our idea of a covering model, in which every event is part of some End (goal) event.

1.3.2.2. Extended Discourse

While a number of researchers [Sidner & Israel 81, Carberry 83, Grosz & Sidner 87] have begun to extend Allen's work to deal with sequences of utterances,

⁶Allen called such chains of inference "partial plans".

[Litman 84] contains the most detailed and concrete proposals.

Litman's plan recognition system includes a set of domain-specific plan schemas, a set of domain-independent *meta-plan* schemas, and an *incremental recognition* algorithm. Meta-plans are plans which can take other plans as arguments. They include, for example, a plan to help a hearer identify a parameter which appears in another plan (IDENTIFY-PARAMETER), and a plan to insert a repair step into a plan which would otherwise fail (CORRECT-PLAN). Litman's work is notable in providing a deterministic, highly constrained recognition algorithm, and in cleanly accounting for plan suspension and resumption.

The plan recognition algorithm constructs a *stack* of partially-recognized plans. When it observes an action, the system attempts to attach it somewhere on the stack, according to the following *preferences*:

1. Attach (as a substep) to the plan on the top of the stack;
2. Attach to a new meta-plan, which refers to a plan somewhere in the stack, and push that meta-plan onto the stack;
3. Attach to a new meta-plan, which refers to some other new plan. If that other plan is also a meta-plan, construct a plan for it to refer to, and so on, until a domain-specific plan is reached. Push everything onto the stack, with the domain-specific plan on the bottom and the original meta-plan on top.

Attaching an observed act to a partial plan may require that certain equalities be *assumed* to hold between parameters of the act and the plan. Litman calls this *consistency unification*. A similar result is obtained in our framework by different means. As will be described in Chapter 4, the assumption that End *events* are equal can lead to the *conclusion* that certain parameters of substeps of the event are equal.

Both Litman's algorithm and our own (as will be seen in Chapter 7) rely on constraint propagation to eliminate alternative interpretations. Her system exemplifies a backtrack-free, incremental recognition process, as described in Chapter 5. Our "sticky" incremental recognition theory is a very crude approximation to Litman's.

A significant difference between her work and ours is our treatment of disjunction. Our theory justifies inference through disjunctions, allowing disjunctions to multiply⁷ in the final conclusion. Litman argues that discourse should provide enough lin-

⁷Although in practice disjunctions often collapse again by abstraction, as will be seen.

guistic *clues* such as intonation, gesture, and keywords to eliminate most disjunctions. Following [Sidner 85], Litman claims that if the interpretation remains ambiguous, one should halt inference and simply wait for further utterances.

1.3.2.3. Cohen and Levesque

Recent work reported in [Cohen & Levesque 80, Cohen & Levesque 87] tries to derive speech act and discourse theory from general principles of rational interaction. The work axiomatizes (part of) the space of possible inferences available to an agent engaged in planning and plan recognition. They have not dealt with selecting between alternative interpretations of an observation.

Cohen & Levesque developed a version of dynamic logic [Harel 79] enriched with a logic of belief [Hintikka 62] to represent actions, beliefs, and intentions. Axioms in this language formally encode rules similar to those of Allen. For example, (one version of) the shared-recognition precondition/action axiom is

```
(imply
  (BMB y x
    (and (or (CAUSE x p (CAN x q))
              (CAUSE x p (CAN y q)))
          (EXPECT y (GOAL x q))
          ¬(GOAL y ¬q)
          (HELPFUL y x)))
    (CAUSE x
      (BMB y x (GOAL x (GOAL y p)))
      (BMB y x (GOAL x
        (GOAL y (FINITELY-WAIT-FOR x q))))
    )
  )
```

The formula (BMB y x ...) means that y believes that it is mutually believed between x and y that ... The axiom states that if y and x mutually believe that p enables q, and y expects that x will eventually want q, and y doesn't want not-q, and y is helpful to x, then *whatever* x does to make it mutually believed that x wants y to want p, will also make it mutually believed that x wants y to achieve q in the future (that is, x won't have to wait forever for q to occur).

Cohen & Levesque have concentrated on getting all the details of the representation of mental attitudes just right, and have deliberately ignored the issue of controlling inference. Our own work has gone to the other extreme, and completely ignored the representation of belief. Indeed, we have not distinguished between what the agent

wants to come about and what *actually will* come about; we've assumed all plans are successful. On the other hand, our system tells us what conclusions are *justified*, rather than merely what conclusions are *possible*. (It is not entirely clear, in any case, what a "possible" conclusion is!) A sophisticated plan recognition system will eventually have to deal with the kinds of representational issues raised by Cohen & Levesque.

1.3.3. Intelligent Computer Environments

A natural application area for the discourse systems discussed above is the human/computer interface itself. Plan recognition is a central component of several programs of research aimed at creating automated consultants, systems which would help a person use a particular, complicated program, or perhaps an entire operating system.

1.3.3.1. The MACSYMA Advisor

One of the earliest automated consultants [Genesereth 79] helped people use MACSYMA, a powerful program for manipulating symbolic equations. Genesereth created a model, MUSER, of how a user typically breaks down a task when using MACSYMA. This model related the task, or plan, structure to the structure of the formulas being manipulated. Plans were represented as procedural nets [Sacerdoti 77], together with input/output links between various steps. The library contained both common plans and common mistakes.

When a user had a problem with MACSYMA, he would invoke the advisor, and tell it both his intended goal and what he had actually done. The advisor then built a possibly "buggy" plan graph which connected the two. The advisor then debugged the plan and told the user what to do.

The advisor used an ordered set of plan recognition rules, which are very similar to those used by Allen and Litman. The rules applied deterministically: the partial plan was expanded only in unambiguous cases. An "escape hatch" existed in that the advisor would ask the user for clarification in case of ambiguity. Genesereth's work raised many issues and techniques which were developed (or rediscovered) by later researchers. Like Litman's system, the consultant provided a "limited" inference mechanism, and could not just chain off in arbitrary directions. He did not, however, formalize the principles above (except in a particular implementation in LISP), or deal with multiple expectations or concurrent multiple plans – issues central to our own work.

1.3.3.2. A Smart Operating System: Plan Parsing

An operating system consultant under development at the University of Massachusetts [Huff & Lesser 82] is notable for dealing with multiple concurrent plans, and for relating plan recognition to *parsing*.

The system tracks user's actions, and allows users to specify high-level commands which are disambiguated by context. A library of operating-system level tasks, such as *compile* or *edit*, is encoded by a set of grammatical rewrite rules, together with a list of *constraints*. The rules employ an extended version of regular expressions.⁸ For example, the plan to update source code is partly described by the rewrite rule:

```
update_source_unit =>
    ((edit compile check_results) |
     (edit compile) |
     (compile check_results) |
     (compile))+
```

Since a programmer may be working on several different projects during the same session, the notion of a regular grammar is extended to that of a *shuffle grammar*. If *e* and *f* are expressions, their shuffle, written *e\$f*, is the set of strings constructed by mixing together a string of *e* with a string of *f*. The interleave of an expression *e*, written *e@*, is the expression shuffled with itself, an arbitrary number of times. For example, the fact that several unrelated programs may be worked on simultaneously is represented by the grammar rule

```
programming_work =>
    (do_programming |
     do_documentation |
     make_errors)@
```

The intelligent interface tries to parse the (partial) input of the user as it is received. It employs heuristics for ordering alternative partial interpretations of an observation when parsing. The heuristics try to "minimize" the amount of mixing performed by the shuffle operator, and the number of shuffles invoked by the interleave operator. For example, it prefers the shuffle *eeeeffff* over *eeffeeff*, which is preferred over

⁸A regular expression is a string made up terminal and non-terminal symbols connected by the operators | (or), * (repeat 0 or more times), + (repeat 1 or more times), and blank (concatenation), possibly grouped by parentheses. A rewrite rule specifies that a non-terminal symbol on its left-hand side may be replaced by the expression on its right.

efefefef. Likewise for interleave, s is preferred over s\$\$s, which is preferred over s\$\$s\$. The heuristics include:

1. Prefer (linking an observation to) an existing plan instantiation over creating a new instantiation.
2. Prefer a new related instantiation to a new unrelated one.
3. If alternative interpretations both appear in the same higher-level containing plan, prefer the interpretation which appears first in that higher-level plan.

Huff & Lesser's parsing heuristics yield conclusions similar to those obtained by the incremental recognition theories discussed in Chapter 5. The plan recognition algorithms described in Chapter 7 have a strong flavor of parsing, despite their origins in logical inference. Just as parsing can be viewed as logical inference⁹, specialized inference can be cast as parsing. But there are at least two crucial differences between our work and Huff & Lesser's. First, we treat all temporal orderings between the components of an event as constraints. Any temporal relation may be specified: for example, one step may be during another, times may overlap, or one step may be required to occur *either* before *or* after another (without specifying which alternative). Second, our algorithms correspond to a particular model theory, while Huff & Lesser's are admittedly ad hoc. The inclusion of the shuffle operator, and the need to reach conclusions before the end of a sequence of observations, prevents Huff & Lesser from using any standard, well-understood parsing algorithms.

1.4. Related Work on Medical Diagnosis

One of the most active areas of applied research in A.I. has been in the area of medical diagnosis. Diagnosis and plan recognition are similar kinds of high-level recognition problems. (In fact, an early paper by [Pople 73], who was later to create the INTERNIST diagnostic system, explicitly made the connection.) In both cases one needs to find the best explanation for some phenomenon, using a hierarchically structured body of domain-specific knowledge. The vocabulary of events can be mapped to one appropriate for diagnosis in a straightforward way. (See Chapter 2 for the details of the Event vocabulary.) Events are replaced by *pathological states* of a patient. An *abstraction* hierarchy over pathological states is known as a *nosology*. The

⁹See any of the work on parsing with Prolog, such as [Pereira & Warren 80].

dealt with this issue, yet it is critical if we are to build systems which can *actively* engage in a discourse or other task involving plan recognition.

1.4.1. INTERNIST and CADUCEUS

One of the best-known diagnostic systems is INTERNIST, an ambitious program which aimed to cover almost the entire field of internal medicine. The first version of INTERIST employed a causal hierarchy, and tried to chain from a group of observed symptoms to one or more specific disease entities. The succeeding versions of the program, INTERNIST-II and ultimately CADUCEUS, employed multiple abstraction (nosology) hierarchies as well. A detailed description of the project appears in [Pople 82].

A *task* for CADUCEUS was a data structure that explained a set of observations by causally linking them to one or more pathological states. The pathological states did not have to be ultimate, specific disease entities; they could be any states within the taxonomy. The task could contain certain kinds of *disjunctions*. For example, a task could link symptom A to causes B, C, and D, meaning that *either* B caused A, *or* C caused A, *or* D caused A. Such a disjunction was called a *differential diagnosis set*. The differential diagnosis sets could *collapse* within the task. For example, all of B, C, and D could be linked by a *subclassification* arc to a more general disease type E.

CADUCEUS created a task for each observed symptom, and then performed a heuristic search to attempt to combine the tasks into a smaller set. During the search, CADUCEUS could interactively prompt for information that could would be used to reduce differential diagnosis sets. An example of a operator that CADUCEUS could apply to two tasks in order to combine them is *Combined Causal/Subclassification*:¹⁰

Where *P* and *Q* are descriptors in tasks *T*₁ and *T*₂ respectively:

IF *P* is (directly) caused by *S*₁ or *S*₂ or ... *S*_{*n*} AND
 *S*₁, *S*₂, ... *S*_{*n*} all (directly) sub-classify *Q* THEN
 Link *T*₁ and *T*₂ by adding caused-by arcs from *P*
 to each of *S*₁, *S*₂, ... *S*_{*n*}, and sub-classification arcs
 from each of *S*₁, *S*₂, ... *S*_{*n*} to *Q*

In Chapter 7, our algorithm for plan recognition employs a data structure called an "e-graph", which links a set of observed events to some End event. E-graphs are

¹⁰The notation here is our own, not Pople's.

decomposition hierarchy corresponds to a *causation* hierarchy. If pathological state A always causes pathological state B, then B acts as a component of A. If only certain cases of A cause B, then one can introduce a specialization of A that has component B. The most basic specializations of End (unexplainable) events correspond to *specific disease entities*, while states which can be directly observed are *symptoms*. (Note that a symptom may also cause other other states: e.g., high blood pressure can be directly measured, and it can cause a heart attack.)

The pattern of inference in plan recognition and diagnosis is similar as well. Each symptom invokes a number of different diseases to consider, just as each observed event in our framework c-entails the disjunction of its uses. Once several findings are obtained, the diagnostician attempts to find a small set of diseases which accounts for, or covers, all the findings. This step corresponds to the minimization of End events in mc-entailment. A general medical diagnosis system must deal with patients suffering from multiple diseases; our plan recognition framework was designed to account for multiple concurrently executing plans. Finally, our work departs from previous work in plan recognition by explicitly dealing with disjunctive conclusions, which are winnowed down by obtaining more observations. These disjunctive sets correspond to the *differential diagnosis* sets which play a central role in medical reasoning.

There are some significant differences between our framework and those used in medical expert systems. In plan recognition it is necessary to distinguish the different roles that one event could play as a component of another, because a plan may have several different steps of the same type. One would need to distinguish the instance of the "meet with committee" event which *initiates* the plan to "obtain PhD degree" from the one which *terminates* it. Medical systems simply have undifferentiated "causes" links from diseases to symptoms. They make the assumption that a disease can't cause two different instances of the same symptom. In addition, the role of *time* is critical in plan recognition, but usually ignored in medical systems. But future medical systems may need to eliminate both differences. It certainly could be the case that a pathological state causes two different instances of the same *abstract* symptom. Thus AIDS often causes multiple different kinds of cancer in the same patient. [Patil, Szolovits, & Schwartz 82] suggests that diagnostic systems should incorporate a model of the time course of a disease, and the difference symptoms which are manifest at each stage.

Much of the research effort in medical expert systems has been on discovery procedures. The diagnostic system must take an active role, asking questions in order to narrow down the set of possible diseases. None of the work in plan recognition has

similar in structure to CADUCEUS's tasks. But because our e-graphs always contain an End node, two e-graphs can be combined by simply performing a top-down match starting at the End nodes. The resulting e-graph encodes *all* possibly ways of combining the original graphs. CADUCEUS performs only a heuristic search, and therefore could not guarantee completeness. There is no straightforward way to tell when all possible combinations have been found, or whether the heuristics need to run longer. The particular set of combining operators are (Pople admits) rather arbitrary and incomplete.

An interesting feature incorporated in CADUCEUS that our framework lacks are *planning links*, which connect one state to another if *some* specialization of the latter can cause the former. [Pople 82] shows how such links can be very useful for heuristically guiding search.

1.4.2. A Set Covering Model

Recent work by [Reggia, Nau, & Wang 83] has proposed that medical diagnosis can be viewed as a set covering problem. Each disease corresponds to the set of its symptoms, and the diagnostic task is to find a minimum cover of a set of observed symptoms.

While this idea has long been implicit in diagnostic systems such as INTERNIST, Reggia & Nau's work is unique in providing a provably correct and complete algorithm which not only finds all minimum covers, but also indicates when and how the diagnostician should request additional findings. The system is particularly strong in dealing with multiple simultaneous diseases.

Reggia & Nau have not yet expanded the work to cover many-level causal or abstraction hierarchies, although they mention that as a topic for future research. The lack of abstraction lets them stay within a purely propositional framework, which greatly simplifies the algorithms. Once causal and abstraction hierarchies are added, their framework and algorithms may prove to be very similar to our own.

1.5. Related Work on Non-Deductive Inference

Most recognition problems cannot be formalized entirely within a deductive system – unless one uses it as a meta-language to talk *about* a different, non-deductive

system.¹¹ This section reviews the major formal systems of non-deductive inference used within A.I., concluding with *circumscription*, the system most closely related to our own work.

1.5.1. Probability Theory

Probability theory provides the basis for non-deductive inference in practically all the sciences *outside* of A.I.. Its success should at least cause the computer scientist to hesitate a bit before devising a new sort of theory. Indeed, a growing number of researchers are basing systems for such tasks as vision or medical diagnosis on classical, Bayesian, or Dempster-Shafer¹² theories of probability. In this section we will describe (part of) plan recognition in terms of very elementary statistics. Our own work in plan recognition, and much of that cited above, can be viewed a method of performing certain steps in the probabilistic inference. A logical theory of plan recognition also addresses issues about which classical probability theory has little to say: most importantly, how hypotheses should be *constructed*, and when a likely statement should be *accepted* as fact.

Let A_1, A_2 , etc. be various directly-observable and executable actions. A countably infinite number of plans, labeled P_1, P_2 , etc. can be constructed from these actions. Each *basic hypothesis* B_i is of the form, "the agent intends to perform plan P_i ." A *composite hypothesis* H is a boolean combination of basic hypotheses. For now, we will only consider composite hypotheses which are *conjunctions* of basic hypotheses.

What is the general form of a plan recognition problem? One might try to determine the most likely composite hypothesis. There is only a trivial answer to this problem: the empty hypothesis, that makes no assertion about the world. We cannot state a plan recognition problem as a crisp decision problem. The best one can do is to try to determine the sets of plans (composite hypotheses) which are "most likely" on the basis of the observed actions. (Later on we'll mention some of the problems in determining the exact point at which a hypothesis is likely enough to be accepted.)

Where P is the probability function, and A is the set of observed actions, the goal is find those H for which $P(H | A)$ is large. By Bayes' theorem,

¹¹ For example, [Kyburg 74] uses first-order logic as a meta-language to axiomatize a system of probabilistic inference.

¹²[Dempster & Shafer 76]

$$P(H | A) = \frac{P(A | H) P(H)}{P(A)}$$

Given a fixed set of observations A and certain simplifying assumptions, it is possible to determine the *relative* probabilities of various candidate hypotheses. Assume that every plan (every B_i) has the same prior probability, and exactly one possible *decomposition* (that is, a breakdown into actions). Begin by considering the case where the actor performs exactly one plan -- that is, the B_i are disjoint. When calculating relative probabilities, the factors $P(A)$ and $P(H)$ can be ignored; the first because A is fixed, the second because every candidate H must be a basic hypothesis. The conditional probability of the observations given the hypothesis is simply:

$$P(A | H) = 1 \text{ if } H \equiv B_i \wedge (\forall A_j \in A) . \text{ component-of}(A_j, P_i) \\ = 0 \text{ otherwise}$$

So the most likely hypotheses are simply those which incorporate all the observations. But since plans are parameterized, there are an infinite number of candidate H 's to consider. We need some way of searching this huge space. Classical statistics seems to have little help to offer; statistical theory has concentrated on the special case where the various H are of the *same* known form (e.g. the percentage of red balls in the urn) parameterized only by some numerical coefficients. But here we must deal with a large number of fundamentally different plans which take different discrete parameters. (The minimal model construction described in this thesis corresponds to this search.)

Relax the assumption that the B_i are disjoint. Now it is harder to compute relative values for $P(H | A)$. First, it is no longer reasonable to assign the same value to all $P(H)$ -- particularly since some composite hypotheses may strictly entail other composite hypotheses. Second, the conditional probability of the observed actions given the candidate hypothesis is much less constrained. Without making additional assumptions, all one may conclude is that

$$P(A | H) = 1 \text{ if } (\forall A_j \in A) (\exists B_i) . H \vdash B_i \wedge \text{ component-of}(A_j, P_i) \\ < 1 \text{ otherwise}$$

If one is comfortable assuming that the B_i are *independent*, so that

$$P(B_1 \wedge B_2 \wedge \dots \wedge B_n) = P(B_1) P(B_2) \dots P(B_n)$$

then note the following: if hypotheses H and H' contain the same number of conjuncts, and if H accounts for all of A but H' does not, then $P(H | A) > P(H' | A)$. Furthermore, if H and H' both account for A , the smaller hypothesis has the higher probability. Our framework for plan recognition has similar characteristics.

We have suggested that in plan recognition it is appropriate to invoke Occam's razor: to prefer explanations which only require the actor to have as few unrelated intentions at a time as possible. This condition does not *quite* fall out of the probabilistic account so far, even assuming the independence of the basic hypotheses: it may give a high probability to an H which does not quite account for all of A , but would have to be greatly expanded to account for the remainder. This can be partially remedied by constraining the conditional probability function $P(A | H)$ to be very small for such an H .

$$P(A | H) \ll P(B_i) \\ \text{if } (\exists A_j \in A) (\forall B_i). H \vdash B_i \supset \neg \text{component-of}(A_j, P_i)$$

A further complication awaits in the wings: it is not adequate to only consider conjunctive composite hypotheses. Much of this thesis will be concerned with abstract event (plan) types. An abstract plan stands for the class of its specializations; that is, it is logically equivalent to the *disjunction* of all the different ways it which it can be fully decomposed with constants assigned to all its parameters. The likelihood that the agent is performing an abstract plan is the probability assigned to the corresponding composite hypothesis, which may be equivalent to an *arbitrary boolean combination* of basic hypotheses. This more general formulation compounds the size of space of hypotheses.

In the end, then, probability gives a way to compare the relative merits of alternative hypotheses about an actors intentions, but does not automatically yield a way of *generating* or *selecting* hypotheses to compare. We don't seem to need any of the high-powered mathematical techniques statisticians have developed. And still the problem of deciding what likely statements to *accept* as true remains.

In our model-theoretic treatment of plan recognition, the statements that the observer should accept are simply those which are valid in the minimal model construction. Using probabilities, one needs some sort of rule of acceptance, or at the least be prepared to deal with all the complexities of allowing an agent to hold *degrees of belief*. [Kyburg 74] deals in detail with the complexities of both these problems. The present situation appears particularly difficult, because we have not come up with absolute

probabilities, but only a method for ordering the relative probabilities of certain statements. Thus no simple rule -- such as "accept H if $P(H) > .95$ " will do.

1.5.2. Default Logic

Much everyday reasoning employs "default rules", that allow conclusions to be drawn if there is no evidence to the contrary. For example, if all one knows about Tweety is that he is a bird, then one may conclude *by default* that Tweety can fly. The additional information that Tweety is a penguin and that penguins don't fly would disallow the previous conclusion. [Reiter 80] developed an extension to first-order logic, called *default logic*, to handle this kind of inference.

How useful are default rules for plan recognition? It is straightforward to cast, for example, Allen's plan recognition rules as default inference rules. Using Reiter's logic, one could write, for example, the *precondition/action* rule as:

$$\frac{\text{Want}(\text{agt}, P) \wedge \text{precondition-of}(P, \text{act}) : M \text{ Want}(\text{agt}, \text{act})}{\text{Want}(\text{agt}, \text{act})}$$

This says that if an agent wants a precondition of an act to hold, and it is consistent that (M) he wants the act, then conclude that he wants the act. But little has been gained from the use of default logic. Default logic has the property that given a set of facts and set of rules, one may reach different and perhaps mutually contradictory sets of conclusions, depending on the order in which one applies the default rules. The logic insures that each set of conclusions, or *extension*, is internally consistent, but gives no way choosing between them. As a basis for plan recognition, then, default logic suffers the same criticisms as the dynamic logic framework of Cohen & Levesque: too much of the problem remains hidden in the strategy which orders the application of various rules of inference.

1.5.3. Circumscription

Circumscription is a specialized form of non-monotonic inference developed by McCarthy to handle the "qualification problem" in planning. McCarthy wanted to be able to formally state that "the objects that can be shown to have a certain property P by reasoning from certain facts A are all the objects that satisfy P." [McCarthy 80] For example, one might have a description of a bunch of blocks on a table, and want to synthesize a plan to build a certain kind of tower. It might be necessary to pick up a

block B, which can only be performed if B is clear. If one cannot *prove* that there is anything on top of B, then, in general, one wants to be able to conclude, by circumscribing the predicate on, that *nothing* is on top of B.

The *circumscription* of a predicate (or set of predicates) relative to a body of knowledge is a sentence of second-order logic which involves the entire collection of facts at hand. The following formulation is drawn from [Lifschitz 84]. Let $S[\pi]$ be a sentence containing the list of predicates π . The expression $S[\sigma]$ is the sentence obtained by rewriting S with each member of π replaced by the corresponding member of σ . The expression $\sigma \leq \pi$ abbreviates the formula stating that the extension of each predicate in σ is a subset of the extension of the corresponding predicate in π ; that is

$$(\forall x . \sigma_1(x) \supset \pi_1(x)) \wedge \dots \wedge (\forall x . \sigma_n(x) \supset \pi_n(x))$$

where each x is a list of variables of the proper arity to serve as arguments to each σ_i . The circumscription of π relative to S , written $\text{Circum}(S[\pi], \pi)$, is the second-order formula

$$S[\pi] \wedge \forall \sigma . (S[\sigma] \wedge \sigma \leq \pi) \supset \pi \leq \sigma$$

Chapter 4 shows how this formula is generalized to allow other symbols to *vary* during the minimization, as described in [McCarthy 84].

Consider the following example. Let $S[p] = p(A) \vee p(B)$, and compute the circumscription of p relative to S . This is:

$$(p(A) \vee p(B)) \wedge \forall \sigma . ((\sigma(A) \vee \sigma(B)) \wedge \sigma \leq p) \supset p \leq \sigma$$

Expanding the $\leq$ abbreviation gives

$$\begin{aligned} & (p(A) \vee p(B)) \wedge \\ & \forall \sigma . ((\sigma(A) \vee \sigma(B)) \wedge \forall x . \sigma(x) \supset p(x)) \supset \\ & \quad \forall x . p(x) \supset \sigma(x) \end{aligned}$$

Now let us investigate what can be concluded from this statement. Instantiate the predicate variable σ with the lambda-expression $\lambda(x) . x=A$. (That is, σ is the predicate which is true of A and nothing else.) This yields

$$\begin{aligned}
& (p(A) \vee p(B)) \wedge \\
& ((A=A) \vee A=B) \wedge \forall x . x=A \supset p(x) \supset \\
& \quad \forall x . p(x) \supset x=A
\end{aligned}$$

This statement reduces to

$$\begin{aligned}
& (p(A) \vee p(B)) \wedge \\
& p(A) \supset (\forall x . p(x) \supset x=A)
\end{aligned}$$

Instantiate the circumscription formula a second time with the expression $\lambda(x).x=B$. The result is

$$\begin{aligned}
& (p(A) \vee p(B)) \wedge \\
& p(B) \supset (\forall x . p(x) \supset x=B)
\end{aligned}$$

Thus the final conclusion is that one of A or B is a p, and there is only one p thing. (The conclusion is not $p(A)$ exclusive-or $p(B)$ because of the possibility that $A=B$.)

$$(\forall x . p(x) \supset x=A) \vee (\forall x . p(x) \supset x=B)$$

This example illustrates the major stumbling block to the use of circumscription. There is no general *mechanical* way of determining how to instantiate the predicate parameters in the second-order formula. The work in this thesis overcomes this problem for a particular class of circumscriptions, and gives simple syntactic rules for computing a *first-order* version of the circumscription formula.

A *model* of a set of statements is *minimal* in π if there is no other model of those statements which is identical, except that π holds of some things in the first model but not in the second.¹³ Section 3.1 formally defines this notion. [Etherington 86] includes proofs that the circumscription of a predicate relative to a formula is true in all models minimal in the predicate. The story of the completeness of circumscription relative to this model theory is a bit complicated. The notion of a minimal model is powerful enough to capture the standard model of arithmetic, which cannot be axiomatized [Davis 80]. While all *strong* second-order models of the circumscription formula are

¹³McCarthy's original version of circumscription did not make reference to a particular set of predicates to be minimized. Instead, the entire *domain* of the model was minimized; thus, a minimal model of formula was one which had no submodels of the formula. [Etherington 86] calls this version of circumscription *domain circumscription*, and has shown that it is *not* comparable in expressive power to predicate circumscription. In this thesis, the expression "minimal model" always means "minimal in some predicate".

minimal models, it is not necessarily the case that all *weak* second-order models are minimal. Another way of stating this is that the circumscription formula is *not* always complete if one views it as a *schema* for recursively generating a set of first-order statements.¹⁴ [Minker & Perlis 85] show that circumscription *is* complete in this sense if in all models the circumscribed predicate has a finite extension. Chapter 3 presents a proof that formulas computed by our closure function *cl* are *complete* for the minimal model semantics.

¹⁴McCarthy's early versions of circumscription viewed the formula in just this way, as a first-order schema, rather than as a true second-order statement.

Chapter 2

Representing Event Hierarchies

2.1. Language

The representation language is first-order predicate calculus with equality. A model provides an interpretation of the language, mapping terms to individuals, functions to mappings from tuples of individuals to individuals, and predicates to sets of tuples of individuals. If M is a model, then this mapping can be made explicit by applying M to a term, function, or predicate. For example, for any model M :

Loves(Sister(Joe),Bill) is true in M if and only if
 $(M[\text{Sister}](M[\text{Joe}]), M[\text{Bill}]) \in M[\text{Loves}]$

Meta-variables (not part of the language) which stand for domain individuals begin with a colon. Thus, one may write:

Let $:C$ be an event token in $\text{Domain}(M)$.

Models map free variables in sentences to individuals. We write $M\{x/:C\}$ to mean the model which is just like M , except that variable x is mapped to individual $:C$. Quantification is defined as follows:

$\exists x . p$ is true in M if and only if
 there exists $:C \in \text{Domain}(M)$ such that p is true in $M\{x/:C\}$
 $\forall x . p$ is true in M if and only if
 $\neg \exists x . \neg p$ is true in M

The propositional connectives are semantically interpreted in the usual way.

Certain finite models can be completely specified by listing all the positive atoms which hold in the model, other than trivial instances of equality (i.e., of the form $a=a$). For example, a model in which the only predicate with a non-empty extension is *Fat*, which has two elements, may be specified by writing:

{ *Fat*(A), *Fat*(B) }

FOL proofs in this thesis use natural deduction, freely appealing to obvious lemmas and transformations. It is convenient to distinguish a set of constant symbols

called *individual parameters*, or just parameters, for use in the deductive rule of existential elimination. The rule allows one to replace an existentially-quantified variable by a parameter which appears at no earlier point in the proof. Parameters are distinguished by the prefix *. Technically, no parameters may appear in the final step of the proof: they must be replaced again by existentially-quantified variables (or eliminated by other means). This final step is omitted when it is obvious how it should be done. For example, we may write

The system concludes $E1(*C) \vee E2(*C)$.

instead of

The system concludes $\exists x . E1(x) \vee E2(x)$.

2.2. Representation of Time, Properties, and Events

Most formal work on representing action has relied on the situation calculus or dynamic logic [Harel 79]. While these formalisms are convenient for planning, they prove awkward for plan recognition: it is impossible (without extreme convolutions; see [Cohen 84]) to state that some particular action *actually occurred* at a particular time. We therefore adopt a "reified" representation of time and events.

Time is linear, and time *intervals* are individuals, each pair related by one of Allen's interval logic relations: Before, Meets, Overlaps, etc. [Allen 83b]. The names of several relations may be written in place of a predicate, in order to stand for the disjunction of those relations, each applied to the same argument pair. For example, the expression

BeforeMeets(T1,T2)

is an abbreviation for the formula

Before(T1,T2) $\vee$ Meets(T1,T2)

As a special case, the predicates **Within** and **Disjoint** abbreviate common disjunctions:

Within(T1,T2) $\equiv$
Starts(T1,T2) $\vee$ During(T1,T2) $\vee$ Finishes(T1,T2)

$\text{Disjoint}(T1, T2) \equiv$

$\text{Before}(T1, T2) \vee \text{Meets}(T1, T2) \vee \text{MetBy}(T1, T2) \vee \text{After}(T1, T2)$

Event *tokens* are also individuals, and event *types* are represented by unary predicates. All event tokens are real; there are no imaginary or "possible" event tokens. Various functions on event tokens, called *roles*, yield parameters of the event. Role functions include the event's *agent* and *time*. For example, the formula

$\text{ReadBook}(C) \wedge \text{agent}(C) = \text{Fred} \wedge \text{time}(C) = T2$

may be used to represent the fact that an instance of booking reading occurs; the agent of the reading is Fred; and the time of the reading is (the interval) T2.

The reader may wonder why we have not used functions to *construct* event tokens from their parameters. For instance, one might want to specify the above action as:

$\text{ReadBook}(\text{Fred}, T2)$

Indeed, many researchers have taken the line that an event type, together with its parameters and time of occurrence completely specifies an event token [Pollack 86, Allen 84]. There are two objections to this alternative. First is that it is not always possible to specify *all* the parameters of an event beforehand. Should ReadBook also have a parameter for the thing being read? What about whether the agent is skimming or reading carefully? And so on.¹ A second problem occurs when we allow very abstract event types, like "ReadSomething", which holds of *any* reading event, or "GainInformation", which holds of any instance of reading, watching television, listening to the radio, etc. It is possible for several *distinct* instances of the same abstract event type to occur *simultaneously*. Thus it is not sufficient to only represent event types and times; explicit event tokens are needed as well.

Timeless propositions are called *facts*. Facts are represented by ordinary predicates. For example, the fact that John is a human may be represented by the formula

$\text{Human}(\text{John})$

The predicate Holds relates a time-dependent *property* and a time interval over which the property holds. Properties are individuals, and are therefore represented by terms. For example, the fact that John is unhappy at time T1 may be represented by the for-

¹This is the same problem people working on case grammar [Fillmore 68] ran into.

mula

Holds(unhappy(John),T1)

The term unhappy(John) denotes the proposition that John is unhappy. Every distinct property term represents a distinct property. This is enforced by a set of *property identity axioms*. The axioms take the following form.

For all distinct $\rho, \sigma \in \text{Property Terms}$:

$$\forall x_1, \dots, x_n . \rho(x_1, \dots, x_n) \neq \sigma(x_1, \dots, x_n)$$

$$\begin{aligned} \forall x_1, \dots, x_n, y_1, \dots, y_n . \rho(x_1, \dots, x_n) = \rho(y_1, \dots, y_n) \supset \\ x_1 = y_1 \wedge \dots \wedge x_n = y_n \end{aligned}$$

All properties are dense: if one holds over an interval, then it holds over all subintervals of that interval. The predicate Never holds of a property and a time when the property holds over no subinterval of the time. The following axioms capture these constraints:

$$\forall p, t_1, t_2 . \text{Holds}(p, t_1) \wedge \text{Within}(t_2, t_1) \supset \text{Holds}(p, t_2)$$

$$\forall p, t_1 . \text{Never}(p, t_1) \equiv (\forall t_2 . \text{Holds}(p, t_2) \supset \text{Disjoint}(t_1, t_2))$$

In addition to the relations between individuals, it is sometimes convenient to talk about relations between predicates. These meta-relations, such as "abstracts" and "component", are not part of the logic itself.

2.3. The Event Hierarchy

An event hierarchy is a collection of restricted-form axioms, and may be viewed as a logical encoding of a *semantic network*, as in [Hayes 85]. These axioms represent the abstraction and decomposition relations between event types. An event hierarchy H contains the following parts, H_E , H_A , H_{EB} , H_D , and H_G :

- H_E is the set of unary event type predicates. H_E contains the distinguished predicates AnyEvent and End.

- H_A is the set of abstraction axioms, each of the form:

$$\forall x . E_1(x) \supset E_2(x)$$

for some $E_1, E_2 \in H_E$. In this case we say that E_2 *directly abstracts* E_1 . The transitive closure of direct abstraction is abstraction; and the fact that E_2 is the same as or abstracts E_1 is written E_2 abstracts* E_1 . AnyEvent abstracts* all event types.

• H_{EB} is the set of basic event type predicates, those members of H_E which do not abstract any other event type.

• H_D is the set of decomposition axioms, each of the form:

$$\forall x . E_0(x) \supset E_1(f_1(x)) \wedge E_2(f_2(x)) \wedge \dots \wedge E_n(f_n(x)) \wedge \kappa$$

where $E_0, \dots, E_n \in H_E$; $f_1, \dots, f_n$ are role functions; and κ is a subformula containing no member of H_E . The formula κ describes the *constraints* on E_0 . E_1 through E_n are called *direct components* of E_0 . Sometimes we refer to a component by the name of its role function; e.g., E_1 is the f_1 direct-component of E_0 . The type End never appears as a direct component of another type; nor does any type which End abstracts.

• H_G is the set of general axioms, those which do not contain any member of H_E . H_G includes the axioms for the temporal interval relations; the density axioms for Holds and Never; the property identity axioms; as well as any other facts not specifically relating to events.

2.4. Components of Event Tokens

The component relation may be applied to event tokens in a model M as follows. Suppose $:C_i$ and $:C_0$ are event tokens. Then $:C_i$ is a direct component of $:C_0$ in M if and only if

(i) there are event types E_i and E_0 , such that $:C_i \in M[E_i]$ and $:C_0 \in M[E_0]$

(ii) H_D contains an axiom of the form:

$$\forall x . E_0(x) \supset E_1(f_1(x)) \wedge \dots \wedge E_i(f_i(x)) \wedge \dots \wedge E_n(f_n(x)) \wedge \kappa$$

(iii) $:C_i = M[f_i](:C_0)$

The component relation is the transitive closure of the direct component relation, and the fact that $:C_n$ is either the same as or a component of $:C_0$ is written $:C_n$ is a component* of $:C_0$.

Note that the component relation over event tokens does not correspond to the transitive closure of the direct-component meta-relation over event types. This is due to the fact that a token may be of more than one type.

2.5. Acyclic Hierarchies & Compatible Types

An *acyclic* hierarchy is one that can be exhaustively searched in finite time, and is formally defined as follows. Two event predicates E_1 and E_2 are *compatible* if there is an event type E_3 such that both E_1 and E_2 abstract* E_3 . A hierarchy is acyclic if it contains no series of event predicates $E_1, E_2, \dots, E_n$ such that:

- (i) E_i is a direct component of E_{i+1} for odd i , $1 \leq i \leq n-1$
- (ii) E_j is compatible with E_{j+1} for even j , $2 \leq j \leq n-2$
- (iii) E_n is compatible with E_1

We will consider only acyclic hierarchies in this thesis, although most results should extend to cyclic hierarchies as well.

Roughly speaking, a lexical hierarchy is cyclic if an event token may have an event of the same type as a component. The presence of the abstraction hierarchy, however, makes this definition too generally applicable. All events are of type AnyEvent; therefore any event token will share at least the type AnyEvent with its components. The previous definition avoids this problem by consider only the meta-structure between event type predicates.

2.6. Example: The Cooking World

The actions involved in cooking form a interesting yet tractable domain for planning and plan recognition. The specialization relations between various kinds of foods are mirrored by specialization relations between the actions which create those foods. Decompositions are associated with the act of preparing a type of food, in the

manner in which a recipe spells out the steps in the food's preparation. A good cook stores information at various levels in his or her abstraction hierarchy. For example, one knows certain actions which are needed to create any cream-based sauce, as well as certain conditions (constraints) which must hold during the preparation. The sauce must be stirred constantly, the heat must be moderate, and so on. A specialization of the type cream-sauce, such an Alfredo sauce, adds steps and constraints: e.g., one should slowly stir in grated cheese at a certain point in the recipe.

The cook and the observer have the same knowledge of cooking, a hierarchically-arranged cookbook. (Most real cookbooks are "flat", of course; years of experience are required to induce the hierarchical structure.) Actions of the cook are reported to the observer, who tries to infer what the cook is making. We do not assume that the reports are exhaustive – there may be unobserved actions – although such an assumption could be made, without changing our framework: we would simply add additional observational reports, of the form "nothing else happened" over certain time periods. A cook may prepare several different dishes at the same time, so it is not always possible to assume that all observations are part of the same recipe. Different End events may share steps. For example, the cook may prepare a large batch of tomato sauce, and then use the sauce in two different dishes.

We do *not* aim to capture a lot of domain-specific information about cooking: rather, cooking appears to be one of the most general "toy worlds" one can consider. (For interesting work on *planning* in the cooking world, see [Schmolze 86].) Techniques which work in the cooking world should translate to almost any other domain.

2.6.1. Diagrammatic Form

The following diagram illustrates a very tiny cooking hierarchy. Thick grey arrows denote the abstraction meta-relation, while thin black arrows denote the direct component meta-relation. All event types are abstracted by AnyEvent. As discussed above, Ends are a special kind of event, which are not components of any other event. Here there are two main categories of End events: preparing meals and washing dishes. It is important to understand that the abstraction hierarchy, encoded by the axioms in H_A , and the decomposition hierarchy, encoded by the axioms in H_D , are interrelated but separate. Work on hierarchical planning often confuses these two distinct notions in an action or event hierarchy.

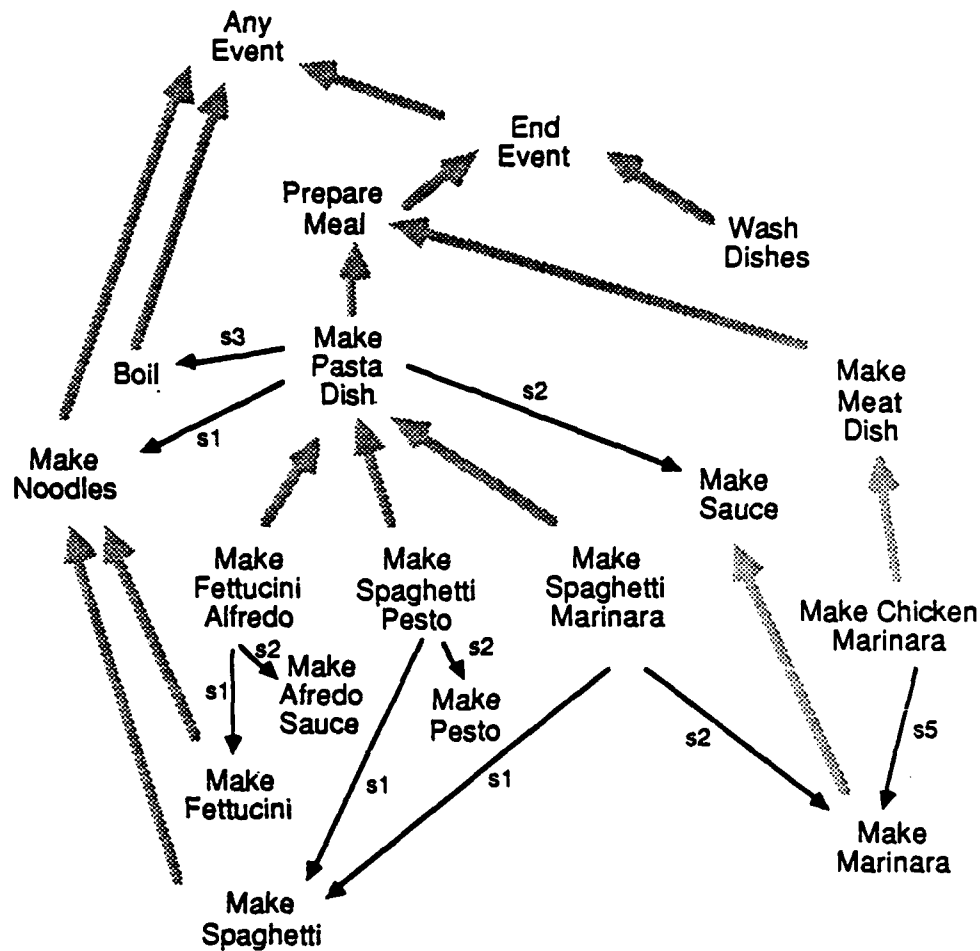


figure 2.1: Cooking Hierarchy

2.6.2. The Abstraction Hierarchy

The diagram suggests some of the elements which make up the lexical hierarchy.

- The set of event types, H_E , includes PrepareMeal, MakeNoodles, MakeFettucini, and so on.
- The abstraction axioms, H_A , assert that every MakeSpaghetti and every MakeFettucini is also a MakeNoodles, that every MakePastaDish is also a PrepareMeal, and so on. A traditional planning system might call MakeSpaghetti and MakeFettucini different *bodies* of the MakeNoodles plan.

$$\forall x . \text{MakeSpaghetti}(x) \supset \text{MakeNoodles}(x)$$
$$\forall x . \text{MakeFettucini}(x) \supset \text{MakeNoodles}(x)$$
$$\forall x . \text{MakePastaDish}(x) \supset \text{PrepareMeal}(x)$$

• The basic event types, H_{EB} , appear at the bottom of the abstraction (grey) hierarchy. These include the types WashDishes, Boil, MakeSpaghettiMarinara, MakeChickenMarinara, MakeFettucini, MakeSpaghetti, and MakeMarinara. Note that basic event types may have components (but no specializations).

2.6.3. The Decomposition Hierarchy

• The decomposition axioms, H_D , include much information which does not appear in the diagram. These axioms specify the role-functions which link an event to its components, and the constraints which hold between those steps and the event. Following is (an abbreviated version of) the decomposition axiom for the MakePastaDish event. This act includes at least three steps: making noodles, making sauce, and boiling the the noodles. The equality constraints assert, among other things, that the agent of each step is the same as the agent of the overall act; and that the noodles the agent makes (specified by the result role function applied to the MakeNoodles step) are the thing boiled (specified by the input role function applied to the Boil step). Temporal constraints explicitly state the temporal relations between the steps and the MakePastaDish. For example, the time of each step is during the time of the MakePastaDish, and the Boil must follow the MakeNoodles. The constraints in the decomposition include the preconditions and effects of the events. Preconditions for MakePastaDish include that the agent is in the kitchen during the event, and that the agent is dexterous (making pasta by hand is no mean feat!). An effect of the event is that there exists something which is a PastaDish, the result of the event, which is ready to eat during a time period postTime, which immediately follows the time of the cooking event.

$\forall x . \text{MakePastaDish}(x) \supset$

	$\text{MakeNoodles}(\text{step1}(x)) \wedge$
<i>Components</i>	$\text{MakeSauce}(\text{step2}(x)) \wedge$
	$\text{Boil}(\text{step3}(x)) \wedge$
<i>Equality</i>	$\text{agent}(\text{step1}(x)) = \text{agent}(x) \wedge$
<i>Constraints</i>	$\text{result}(\text{step1}(x)) = \text{input}(\text{step3}(x)) \wedge$
<i>Temporal</i>	$\text{During}(\text{time}(\text{step1}(x)), \text{time}(x)) \wedge$
<i>Constraints</i>	$\text{BeforeMeets}(\text{time}(\text{step1}(x)), \text{time}(\text{step3}(x))) \wedge$
	$\text{Overlaps}(\text{time}(x), \text{postTime}(x)) \wedge$
<i>Preconditions</i>	$\text{Holds}(\text{inKitchen}(\text{agent}(x)), \text{time}(x)) \wedge$
	$\text{Dexterous}(\text{agent}(x)) \wedge$
<i>Effects</i>	$\text{Holds}(\text{readyToEat}(\text{result}(x)), \text{postTime}(x)) \wedge$
	$\text{PastaDish}(\text{result}(x))$

Note that the names of the component roles, step1, step2, etc., are arbitrary; they do *not* indicate temporal ordering. The event types which specialize MakePastaDish add additional constraints and steps to its decomposition. For example, the event type MakeSpaghettiMarinara further constrains its decomposition to include MakeSpaghetti (rather than the more generic MakeNoodles) and MakeMarinaraSauce (rather than simply MakeSauce). One could also add completely new steps as well.

$\forall x . \text{MakeSpaghettiMarinara}(x) \supset$

$\text{MakeSpaghetti}(\text{step1}(x)) \wedge$
 $\text{MakeMarinaraSauce}(\text{step2}(x)) \wedge \dots$

2.6.4. Describing Instances of Events

Assertions about particular event instances take the form of the predication of an event type of a constant, conjoined with equality assertions about the roles of the event token, and perhaps a proposition relating the time of the event to that of other events. The English statement, "Joe made the noodles on the table yesterday" may be represented as follows:

$\text{MakeNoodle}(\text{Make33}) \wedge$
 $\text{agent}(\text{Make33}) = \text{Joe} \wedge$
 $\text{result}(\text{Make33}) = \text{Noodles72} \wedge$
 $\text{Holds}(\text{onTable}(\text{Noodles72}), \text{Tnow}) \wedge$
 $\text{During}(\text{time}(\text{Make33}), \text{Tyesterday})$

2.7. Conditional Actions

Plans of action often contain conditional actions. If some condition holds, then the plan (or event type) contains a certain step; otherwise, it contains a different step. At first glance, the form of a lexical hierarchy does not seem to allow for the presence of conditional steps in the decomposition of an event type. Consider the case where if a condition P holds, event type E_0 should contain a step of type E_T ; otherwise, it contains one of type E_F . One may want to write something like the following.

$$\begin{aligned} \forall x . E_0(x) \supset & (P \supset (E_T(s_T(x)) \wedge \kappa_T)) \wedge \\ & (\neg P \supset (E_F(s_F(x)) \wedge \kappa_F)) \wedge \\ & E_2(s_2(x)) \wedge \dots \wedge \kappa \end{aligned}$$

The expressions κ_T and κ_F introduce constraints which only hold on the conditional part of the event. This formula is clearly not of the form specified for axioms in H_D . Does this mean that all the work in this thesis that depends on lexical hierarchies must be redone in order to account for conditional events?

2.7.1. Representing Conditional Actions

Fortunately, the answer is no. The form of a lexical hierarchy is sufficiently general to accommodate conditional events, through the introduction of additional abstract event types. Consider the case above. There are two clear ways of specializing the event type E_0 . In the first category, P holds, and E_0 contains the component E_T . In the second category, P does not hold, and E_0 contains the component E_F . This situation is captured by the following axioms.

$$\forall x . E_0(x) \supset E_2(s_2(x)) \wedge \dots \wedge \kappa$$

$$\forall x . E_{0T}(x) \supset E_0(x)$$

$$\forall x . E_{0F}(x) \supset E_0(x)$$

$$\forall x . E_{0T}(x) \supset E_T(s_T(x)) \wedge \kappa_T \wedge P$$

$$\forall x . E_{0F}(x) \supset E_F(s_F(x)) \wedge \kappa_F \wedge \neg P$$

It is plain that these axioms are of the proper form to appear in H_A and H_D . The process that transformed the first statement into the second set of statements could be re-

peated, in case $E0$ contains more than one conditional expression. In fact, this forms a proof that conditional statements can always be "factored out" of decomposition axioms.

This is not the only possible way of representing conditionals in a lexical hierarchy. Another method would be to replace the conditional expression in the decomposition of $E0$ by a new event type, and then give two specializations for that event type. The result could be as follows.

$$\forall x . E0(x) \supset \begin{array}{l} Ep(s1(x)) \wedge \kappa_P \wedge \\ E2(s2(x)) \wedge \dots \wedge \kappa \end{array}$$

$$\forall x . Ep_T(x) \supset Ep(x)$$

$$\forall x . Ep_F(x) \supset Ep(x)$$

$$\forall x . Ep_T(x) \supset E_T(s_T(x)) \wedge \kappa_T \wedge P$$

$$\forall x . Ep_F(x) \supset E_F(s_F(x)) \wedge \kappa_F \wedge \neg P$$

The new constraint expression κ_P in the first axiom makes $s1(x)$ have all the same roles as x , except for the new role $s1$ itself. This is necessary so the constraint expressions κ_T and κ_F in the final two axioms properly constrain $s_T(x)$ and $s_F(x)$. The advantage of this method is that it results in fewer axioms if the initial decomposition axiom contains many conditional expressions. Where n is the number of conditionals, the former method could introduce as many as $O(2^n)$ axioms, while the latter method only introduces $O(n)$ axioms.

Some question might arise in the case where half the conditional is empty. Suppose that a certain action had to be performed only if P held:

$$\forall x . E0(x) \supset \begin{array}{l} (P \supset (E_T(s_T(x)) \wedge \kappa_T)) \wedge \\ E2(s2(x)) \wedge \dots \wedge \kappa \end{array}$$

Either transformation goes through as before, with the elimination of the atom $E_F(s_F(x))$. One of the axioms generated by the second method, however, looks a bit odd. The decomposition axiom for the event Ep_F contains no event-types in its consequence:

$$\forall x . Ep_f(x) \supset \neg P$$

What is intuitive meaning of the event type Ep_f ? It is a kind of "null event". One will never directly observe an instance of Ep_f ; however, from $\exists x.E0(x)$ and $\neg P$ one can deduce that Ep_f occurred. Unlike some accounts of null actions, we do not insist that a null action occurs over all time periods in which the world is unchanged.

2.7.2. An Example

The **PickUp** action in the cooking world depends critically upon the temperature of the object being picked up. If the object is hot, the agent must wear a mit to avoid being burned. Therefore the plan for picking up an object must begin with the action of conditionally putting on a mit if the object is hot. This can be encoded in an event hierarchy as follows.

$$\begin{aligned} \forall x . \text{PickUp}(x) \supset \\ \text{Grasp}(s2(x)) \wedge \text{object}(x) = \text{object}(s2(x)) \wedge \dots \end{aligned}$$

$$\forall x . \text{PickUpHot}(x) \supset \text{PickUp}(x)$$

$$\forall x . \text{PickUpCool}(x) \supset \text{PickUp}(x)$$

$$\begin{aligned} \forall x . \text{PickUpHot}(x) \supset \\ \text{Holds}(\text{Hot}(\text{object}(x)), \text{time}(x)) \wedge \\ \text{PutOnMit}(s1(x)) \wedge \text{BeforeMeet}(\text{time}(s1(x)), \text{time}(s2(x))) \wedge \dots \end{aligned}$$

$$\begin{aligned} \forall x . \text{PickUpCool}(x) \supset \\ \text{Never}(\text{Hot}(\text{object}(x)), \text{time}(x)) \end{aligned}$$

Every **PickUp** includes grasping the object, lifting it, and so on. **PickUps** are either of hot or cool objects. Picking up a hot object requires a preliminary step of putting on a mit. Picking up a cool object is constrained to occur only when the object is never hot during the time of the **PickUp**.

Chapter 3

Covering Models

We have seen that there are too many models of an event hierarchy to construct a semantic basis for recognition. A technique known as *model minimization* can be used to select a suitable subset, called *covering models*. In a covering model, any non-End event is a component of some End event. Each covering model for an observation serves as an *explanation*, in terms of End events, of the observation. While it would be unwise to arbitrarily adopt a *particular* covering model, it is reasonable to conclude whatever propositions hold in *all* covering models. These propositions are *c-entailed* by the observation. The sequence of model minimizations used to construct the covering models corresponds to a complex application of McCarthy's *predicate circumscription* schema. An easily-computed set of *completeness assumptions* provides a complete proof-theoretic description of the covering models.

Recall that a lexical hierarchy H contains two major parts: an abstraction hierarchy, H_A , and a decomposition hierarchy, H_D . Each of these hierarchies must be strengthened in order to be used for recognition. The abstraction hierarchy is strengthened by *assuming* that there are no event types outside of H_E , and that all abstraction relations between event predicates are derivable from H_A . The decomposition hierarchy is strengthened by *assuming* that non-End events occur only as components of other events.

These assumptions are reasonable because H encodes *all* of our knowledge of events. If the hierarchy is enlarged, the assumptions must be revised. (This thesis does not deal with learning, but may be compatible with various learning strategies. An obvious approach is to try to explain the observations while assuming H is complete; if there are no covering models of small cardinality in End, then relax some of the completeness assumptions.) At the conceptual level one can imagine computing all the completeness assumptions each time the hierarchy is modified. An implementation, of course, could incrementally build structures representing the assumptions as each part of the hierarchy is created.

3.1 Model Minimization

Let M_1 be a member of a class of models μ , and let π be a set of predicates. M_1 is minimal in π among μ if and only if there does not exist any other model M_2 such that:

1. M_2 is a member of μ .
2. M_1 and M_2 have the same domain.
3. M_1 and M_2 agree on the interpretation of all constants, functions, and predicates not in π .
3. The extension of every member of π in M_2 is a subset of the extension of that predicate in M_1 .
5. The extension of some member of π in M_2 is a proper subset of the extension of that predicate in M_1 .

If M_1 fails to be minimal because of such an M_2 , we say that M_2 defeats the candidacy of M_1 .

3.2 Completing the Abstraction Hierarchy

The following conditions incrementally define the A-closed models of H . In each case, M is a model of H_A .

- M is closed under specialization if M is minimal in $H_E - H_{EB}$ among models of H_A . That is, all non-basic event types are minimized.
- M is closed under abstraction if M is minimal in $H_E - \{\text{AnyEvent}\}$ among models of H_A which are closed under specialization.
- Finally, we define M to be an A-closed model of H just in case M is a model of H , and M is also a model of H_A which is closed under abstraction.

The following theorems describe the A-closed models in proof-theoretic terms. Proofs appear in the Appendix. Theorem 3.1 says that the given specializations of each

type are assumed to be exhaustive. Theorem 3.5 shows that we have assumed that all event types are disjoint, unless H explicitly states otherwise. (Disjointedness assumptions are used heavily in recognition, as demonstrated below. We do *not* assert that different event types cannot occur simultaneously; only that a particular *token* cannot be of two non-compatible types.) Theorem 3.7 says that every event is of exactly one basic type. Theorem 3.9 presents a complete axiomatization of the A-closed models.

3.2.1. Theorem 3.1 (Exhaustiveness)

Suppose $\{E_1, E_2, \dots, E_n\}$ are all the predicates directly abstracted by E_0 in H_A . Then the statement:

$$\forall x . E_0(x) \supset (E_1(x) \vee E_2(x) \vee \dots \vee E_n(x))$$

is true in all models of H_A which are closed under specialization. The statement is also true in all A-closed models of H .

3.2.2. Theorem 3.5 (Disjointedness)

If event predicates E_1 and E_2 are not compatible, then the statement:

$$\forall x . \neg E_1(x) \vee \neg E_2(x)$$

is true in all models of H_A which are closed under abstraction. The statement is also true in all A-closed models of H .

3.2.3. Theorem 3.7 (Unique Basic Types)

If M_1 is a model of H_A closed under abstraction containing event token $:C_1$, then there is a unique basic event type E_b such that $:C \in M_1[E_b]$. Any event type which holds of $:C$ abstracts* E_b .

3.2.4. Theorem 3.9 (Abstraction Completeness)

Let EXA be the set of all statements which instantiate Theorem 3.1, and let DJA be the set of all statements which instantiate Theorem 3.5 for a particular H . M_1 is an A-closed model of H if and only if M_1 is a model of $H \cup \text{EXA} \cup \text{DJA}$.

3.3 Completing the Decomposition Hierarchy

Once the abstraction hierarchy has been closed, it is a simple matter to close the decomposition hierarchy, by minimizing the set of non-End event types. C-entailment is then defined in terms of the covering models. Theorem 3.10 says that every non-End event must be a component of some other event, and Theorem 3.11 allows one to infer the disjunction of possible uses of observed event token. By Theorem 3.13 every event is part of an End event, and by 3.14 the upward-inference assumptions exactly axiomatize the covering models. Theorem 3.15 states the obvious corollary that c-entailment is computable. Theorem 3.16 shows that one must consider all the possible abstractions and specializations of an event in order to account for all of its possible uses -- and therefore predicate completion in the style of [Clark 78] does not correspond to decomposition completion.

3.3.1. Definition of Covering Model and C-Entailment

M is a **covering model** of H if M is minimal in $H_E - \{\text{End}\}$ among A -closed models of H . Then Γ **c-entails** Ω , written

$$\Gamma_H \models_c \Omega$$

when Ω holds in all covering models of H in which Γ holds. If Ω holds for any Γ , Ω is **c-valid**.

3.3.2. Theorem 3.10 (No Useless Events)

Let M_1 be a covering model of H , containing event token $:C_1$. Then either $:C_1 \in M_1[\text{End}]$ is true, or there exists some event token $:C_2$ such that $:C_1$ is a direct component of $:C_2$.

3.3.3. Theorem 3.11 (Component/Use)

Let $E \in H_E$, and $\text{Com}(E)$ be the set of event predicates with which E is compatible. Consider all the decomposition axioms in which any element of $\text{Com}(E)$ appears on the right-hand side. The j -th such decomposition axiom has the following form, where E_{ji} is the element of $\text{Com}(E)$:

$$\forall x . E_{j0}(x) \supset E_{j1}(f_{j1}(x)) \wedge \dots \wedge E_{ji}(f_{ji}(x)) \wedge \dots \wedge E_{jn}(f_{jn}(x)) \wedge \kappa$$

Suppose that the series of these axioms, where an axiom is repeated as many times as there are members of $\text{Com}(E)$ in its right-hand side, is of length $m > 0$. Then the following statement is c-valid:

$$\begin{aligned} \forall x . E(x) \supset & \text{End}(x) \vee \\ & (\exists y . E_{1,0}(y) \wedge f_{1i}(y)=x) \vee \\ & (\exists y . E_{2,0}(y) \wedge f_{2i}(y)=x) \vee \\ & \dots \vee \\ & (\exists y . E_{m,0}(y) \wedge f_{mi}(y)=x) \end{aligned}$$

3.3.4. Theorem 3.13 (No Infinite Chains)

If M_1 is a covering model of H such that $C_1 \in M_1[E]$, then there is a $:C_n$ such that $:C_n \in M_1[\text{End}]$ and $:C_1$ is a component* of $:C_n$.

3.3.5. Theorem 3.14 (Decomposition Completeness)

Let CUA be the set of all formulas which instantiate Theorem 3.11 for a particular H . M_1 is a covering model of H if and only if M_1 is a model of $H \cup \text{EXA} \cup \text{DJA} \cup \text{CUA}$.

3.3.6. Theorem 3.15 (Computability of C-Entailment)

There is a computable function cl which maps a hierarchy H into a set of axioms with the property that

$$\Gamma_H \models_c \Omega$$

if and only if

$$\text{cl}(H) \cup \Gamma \models \Omega$$

3.3.7. Theorem 3.16 (Not Predicate Completion)

Theorem 3.11 cannot be strengthened by considering only axioms in which E appears as a component, instead of all ones in which event types compatible with E appear as components.

3.4 Circumscription

Predicate circumscription [McCarthy 84] provides a proof theoretic realization of the model-theoretic minimalization operation used above. Direct use of the circumscription schema is difficult, however. Its most general form is a second-order, rather than a first-order statement. Techniques are known for automatically computing first-order circumscriptions for certain kinds of axiom sets; for example, for horn-clauses [Bossu & Seigel 85], or "separable" databases [Lifschitz 84]. None of these previously known techniques apply in the case under consideration here. Although the original hierarchy may be in horn-clauses, the result of the first minimization is a non-horn set of sentences, so the techniques based on predicate-completion are not applicable. None of the minimizations involve separable sets of predicates. However, the restricted form of lexical hierarchies has allowed us to directly compute a set of first-order statements which characterize all models resulting from a certain sequence of minimizations. This work thus describes a special but useful case in which circumscription can be efficiently computed.

Recall that the circumscription of the set of predicates π over a formula S, written $\text{Circum}(S[\pi], \pi)$, stands for the second-order formula

$$S[\pi] \wedge \forall \sigma . (S[\sigma] \wedge \sigma \leq \pi) \supset \pi \leq \sigma$$

where the expression $\sigma \leq \pi$ abbreviates the formula stating that the extension of each predicate in σ is a subset of the extension of the corresponding predicate in π ; that is:

$$(\forall x . \sigma_1(x) \supset \pi_1(x)) \wedge \dots \wedge (\forall x . \sigma_n(x) \supset \pi_n(x))$$

3.4.1. Theorem 3.17 (C-entailment and Circumscription)

For a given a hierarchy H, a statement Ω is c-entailed by Γ if and only if Ω follows from the following schema:

$$\Gamma \wedge \text{Circum}(H \wedge \text{Circum}(\text{Circum}(H_A, H_E - H_{EB}), \{ \text{AnyEvent} \}), H_E - \{ \text{End} \})$$

Note that the two inner circumscriptions apply only to the abstraction hierarchy, while the final circumscription applies to all of H .

3.5 Example: The Cooking World, Continued

We return to the domain of the cooking, and discuss some of the statements which appear in the closure of the lexical hierarchy. These statements are then used to solve a simple plan recognition problem. The closure function, cl , generates three sets of axioms: the exhaustiveness assumptions (EXA), disjointedness assumptions (DJA), and component/use assumptions (CUA). We consider each set in turn.

3.5.1. Exhaustiveness Assumptions (EXA)

These axioms arise by minimizing non-basic event types. They justify inferences from an abstract type to the disjunctive of its specializations. In the cooking world, EXA includes the assertion that all End events are either instances of preparing meals or washing dishes:

$$\forall x . \text{End}(x) \supset \text{PrepareMeal}(x) \vee \text{CleanHouse}(x)$$

Similarly, all instances of preparing meals are either instances of making a pasta dish or making a meat dish.

$$\forall x . \text{PrepareMeal}(x) \supset \text{MakePastaDish}(x) \vee \text{MakeMeatDish}(x)$$

One such axiom appears for every event type not in H_{EB} .

3.5.2. Disjointedness Assumptions (DJA)

These axioms arise by minimizing all event types other than AnyEvent. This minimizes the set of types of each event token. By EXA, every event is of some basic type; by DJA, it is of no more than one basic type. The final effect of this is that types

are disjoint, unless one abstracts the other, or they abstract a common type. It is important to reiterate that the assumptions do *not* assert that different event types cannot occur simultaneously; only that a particular *token* cannot be of two non-compatible types. The cooking world example includes the assumptions that preparing a meal and cleaning a house are disjoint; making a pasta dish and making a meat dish are disjoint; and so on.

$$\forall x . \neg \text{PrepareMeal}(x) \vee \neg \text{CleanHouse}(x)$$

$$\forall x . \neg \text{MakePastaDish}(x) \vee \neg \text{MakeMeatDish}(x)$$

It is simply to "block" a potential disjointedness assumption by adding a new type. For example, suppose we do not want to make the assumption that pasta dishes and meat dishes are disjoint. Then add a type to the original hierarchy which is abstracted by both; for example, *MakeMeatRavioli*.

adding to H_A

$$\forall x . \text{MakeMeatRavioli}(x) \supset \text{MakePastaDish}(x)$$

$$\forall x . \text{MakeMeatRavioli}(x) \supset \text{MakeMeatDish}(x)$$

would eliminate from DJA

$$\forall x . \neg \text{MakePastaDish}(x) \vee \neg \text{MakeMeatDish}(x)$$

3.5.3. Component/Use Assumptions (CUA)

The most important assumptions for recognition arise from minimizing non-End events. The assumptions in CUA let one infer the disjunction of the possible causes for an event from its occurrence. The axioms take us from an event to an event which has a compatible type as a component. The simplest case is when only a single type could have a particular event as a direct component. For instance, the hierarchy has only a single use for the action Boil, namely *MakePastaDish*.

$$\begin{aligned} \forall x . \text{Boil}(x) \supset \\ \exists y . \text{MakePastaDish}(y) \wedge x = \text{step3}(y) \end{aligned}$$

It is frequently possible to simplify the statements in CUA, by taking advantage of the abstraction axioms. For example, according the rule for construction of CUA, the upward-inference axiom for MakeNoodle is:

$$\begin{aligned} \forall x . \text{MakeNoodle}(x) \supset \\ (\exists y . \text{MakePastaDish}(y) \wedge x = \text{step1}(y)) \vee \\ (\exists y . \text{MakeSpaghettiMarina}(y) \wedge x = \text{step1}(y)) \vee \\ (\exists y . \text{MakeSpaghettiPesto}(y) \wedge x = \text{step1}(y)) \vee \\ (\exists y . \text{MakeFettuciniAlfredo}(y) \wedge x = \text{step1}(y)) \end{aligned}$$

The following axiom is equivalent, in light of H_A .

$$\begin{aligned} \forall x . \text{MakeNoodle}(x) \supset \\ \exists y . \text{MakePastaDish}(y) \wedge M1 = \text{step1}(y) \end{aligned}$$

The simplification could not be made if MakePastaDish did not abstract all of the other event types, MakeSpaghettiMarina, MakeSpaghettiPesto, and MakeFettuciniAlfredo.

Another axiom that will be useful in our examples lets one infer the two known uses for making marinara sauce, namely making spaghetti marinara and making chicken marinara.

$$\begin{aligned} \forall x . \text{MakeMarinara}(x) \supset \\ (\exists y . \text{MakeSpaghettiMarinara}(y) \wedge x = \text{step1}(y)) \vee \\ (\exists y . \text{MakeChickenMarinara}(y) \wedge x = \text{step3}(y)) \end{aligned}$$

This statement has also been simplified. The original form should include the disjunct MakePastaDish, since MakePastaDish has the direct component MakeSauce, which is compatible with MakeMarinara. However, the exhaustiveness assumptions can be used to conclude that any MakePastaDish is either MakeSpaghettiMarinara, MakeSpaghettiPesto, or MakeFettuciniAlfredo. The step1 role in the latter two cases must be filled by an event of type MakePesto or MakeAlfredo respectively. The disjointedness assumptions can then be used to infer that MakeMarinaraSauce, MakePesto, and MakeAlfredo are mutually disjoint. Therefore the possibility that MakeMarinaraSauce could be a component of MakeSpaghettiPesto or MakeFettuciniAlfredo can be eliminated. We will use without further comment simplified versions of these assumptions in CUA.

Finally, the UPA axiom for MakeSauce demonstrates the importance of considering compatible types. The only *direct* use of MakeSauce is MakePastaDish; however, MakeSauce abstracts MakeMarinara, and *that* event type appears in the decomposition of MakeChickenMarinara. Therefore the axiom is:

$$\begin{aligned} \forall x . \text{MakeSauce}(x) \supset \\ (\exists y . \text{MakePastaDish}(y) \wedge x = \text{step1}(y)) \vee \\ (\exists y . \text{MakeChickenMarinara}(y) \wedge x = \text{step5}(y)) \end{aligned}$$

3.5.4. A Simple Recognition Problem

The assumptions justified by c-entailment can be used to solve simple recognition problems. Let us suppose that the observer learns that the cook is either making spaghetti or fettucini. This is not enough information to conclude a particular basic dish is being created. It is safe to conclude, however, that *some* pasta dish is in the works. From this abstract description of the End event in progress, the observer can still make useful predictions. For example, he knows that the agent will eventually begin to boil water. A helpful observer might fetch a pot of water; a enemy observer might shut off the water supply!

Following is a proof sketch. The justification for each step appears in italics.

Observation

MakeFettucini(M1) $\vee$ MakeSpaghetti(M1)

Abstraction

MakeNoodle(M1)

Component/use, Existential Instantiation

MakePastaDish(*I1)

Abstraction

End(*I1)

Decomposition

Boil(step3(*I1)) $\wedge$ After(time(M1), time(step3(*I1)))

The final step is the predication, that a boiling event will occur at some time after the observation. As discussed in Chapter 2, the final logical form should replace the individual parameter *I1 by an existentially-quantified variable, and write

$$\exists y . \text{Boil}(\text{step3}(y)) \wedge \text{After}(\text{time}(\text{M1}), \text{time}(\text{step3}(y)))$$

Chapter 4

Minimum Covering Models

C-entailment does not combine information from several observations. We would like to intersect possible explanations for each event. This is done by selecting covering models which minimize the *number* of end events. Minimization can be understood either as producing a set with as few elements as possible, or as producing a set with no redundant elements. The former sense is called cardinality minimization, and the latter, set minimization. (The use of the word minimization without qualification will always mean set minimization.) The model minimization schemes used in Chapter 3 are all instances of set minimization. The work in this chapter relies on cardinality minimization. We will show that certain cases of *circumscription with variables* correspond to numeric minimization.

4.1. Cardinality Minimization

Let M_1 be a member of a class of models μ , and let π be a predicate. M_1 has minimum cardinality in π among μ if and only if there does not exist any other model M_2 such that:

1. M_2 is a member of μ .
2. The size of the extension of π in M_2 is smaller than the size of the extension of π in M_1 . That is,
$$|M_2[\pi]| < |M_1[\pi]|$$

If M_1 fails to be minimal because of such an M_2 , we say that M_2 defeats the candidacy of M_1 .

Let Γ be any sentence or set of sentences, and H a hierarchy. M is a minimum cover of Γ (relative to H) just in case

1. M is a model of Γ
2. M is a covering model of H
3. M has minimum cardinality in End among covering models of H

Suppose Γ is any sentence. Then P is mc-entailed by Γ , written

$$\Gamma_H \models_{mc} \Omega$$

if Ω holds in all minimum covers of Γ .

The proof-theoretic counterpart of cardinality minimization is to adopt the strongest statement which limits the number of distinct end events. Unfortunately, this step is not always effectively computable (the usual problem with default reasoning). In practice, one makes the strongest assumption possible; forward chains a limited amount; if a contradiction is detected, then makes the next weaker assumption, and repeats.

4.1.1. Theorem 4.1 (Minimum Cardinality Defaults)

Consider the following sequences of statements.

$$MA_0. \quad \forall x. \neg \text{End}(x)$$

$$MA_1. \quad \forall x, y. \text{End}(x) \wedge \text{End}(y) \supset x=y$$

$$MA_2. \quad \forall x, y, z. \text{End}(x) \wedge \text{End}(y) \wedge \text{End}(z) \\ \supset (x=y) \vee (x=z) \vee (y=z)$$

...

The first asserts that no End events exist; the second, no more than one End event exists; the third, no more than two; and so on. Suppose there is a minimum covering model in which the extension of End is finite. Then

$$\Gamma_H \models_{mc} \Omega$$

if and only if

$$\Gamma \cup \text{cl}(H) \cup MA_i \vdash \Omega$$

where i is the smallest integer such that left-hand side of the provability relation is consistent.

4.2. Cardinality Circumscription

At first glance it would seem that the theory of circumscription cannot handle the problem of minimizing the cardinality of the extension of a predicate. The circumscription schema states that there is no predicate which satisfies the axioms for the minimized predicate which has an extension which is a proper subset of that of the predicate. The extension must be minimal, not a minimum.

But consider the role of the non-minimized symbols in the circumscription. In the simplest version of circumscription, used in the previous chapter, these symbols must have the same denotation in comparable models. But the more general theory of circumscription [McCarthy 85] allows one to specify that certain predicates, functions, and/or constants *vary* during the minimization. Models may be comparable even if they do not agree on those symbols. The definition of minimum cardinality above compares models without regard to their agreement on any symbols. One could try to represent this process by circumscribing a predicate where all other symbols are allowed to vary. Surprisingly, this works! Under easily met conditions, minimizing cardinality is equivalent to setwise minimization where all predicates and functions vary.

Why should this be the case? Suppose we have two models, M_1 and M_2 , where the cardinality of the extension of End is larger in M_1 than it is in M_2 , yet the extension of End in M_1 does not contain the extension of End in M_2 . M_1 and M_2 are not comparable. But there must be a model that "looks like" M_2 , which is comparable to M_1 . This model simply swaps the roles played by certain domain elements in all predicates and functions, so that its extension of End is a proper subset of M_1 's. We can precisely define how this can be done.

4.2.1. Circumscription with Variables

The circumscription schema can be modified to allow symbols to vary. The circumscription of π over a formula S where α varies is written as

$$\text{Circum}(S[\pi, \alpha], \pi, \alpha)$$

which abbreviates the second-order formula

$$S[\pi, \alpha] \wedge \forall \sigma, \beta . (S[\sigma, \beta] \wedge \sigma \leq \pi) \supset \pi \leq \sigma$$

In the schema, α stands for either a single symbol or a sequence of symbols.

4.2.2. Theorem 4.2 (Cardinality Circumscription)

Let α include all the predicate, function, and constant symbols in our language other than End. Suppose that all models of H are infinite, and in some model of $\Gamma \cup \text{cl}(H)$, End has a finite extension. If

$$\text{Circum}(\Gamma \cup \text{cl}(H), \{\text{End}\}, \alpha) \vdash \Omega$$

then

$$\Gamma_H \models_{mc} \Omega$$

where $\text{Circum}(\Gamma \cup \text{cl}(H), \{\text{End}\}, \alpha)$ means to circumscribe with α varying. The "if" is strengthened to "if and only if" if it is true that circumscription is complete in this case.

4.2.3. Example of Cardinality Circumscription

Consider the following simple example of circumscribing the cardinality of a predicate. The formula to be circumscribed contains only the predicate End, and three constant symbols. It asserts that either a is End, or both of b and c are.

$$S[\text{End}, a, b, c] = a \neq b \wedge a \neq c \wedge b \neq c \wedge (\text{End}(a) \vee (\text{End}(b) \wedge \text{End}(c)))$$

The circumscription of End in S where the constants do not vary simply strengthens the disjunction to exclusive or.

$$\begin{aligned} \text{Circum}(S[\text{End}, a, b, c], \text{End}) \equiv \\ a \neq b \wedge a \neq c \wedge b \neq c \wedge (\text{End}(a) \oplus (\text{End}(b) \wedge \text{End}(c))) \end{aligned}$$

This is because there is a minimal model where both b and c are End. Now try circumscribing with a, b, and c varying.

$$\begin{aligned} \text{Circum}(S[\text{End}, a, b, c], \text{End}, \{a, b, c\}) \equiv \\ S[\text{End}, a, b, c] \wedge \\ \forall p, x, y, z . (S[p, x, y, z] \wedge \forall w . p(w) \supset \text{End}(w)) \supset \\ \forall w . \text{End}(w) \supset p(w) \end{aligned}$$

Instantiations for p, x, y, and z are chosen.

Let $p = (\lambda u . u = b)$
 $x = b$
 $y = a$
 $z = c$

The instantiated schema becomes:

$S[\text{End}, a, b, c] \wedge$
 $(S[(\lambda u . u = b), b, a, c] \wedge \text{End}(b)) \supset \forall w . \text{End}(w) \supset w = b$

Expanding $S[(\lambda u . u = b), b, a, c]$:

$S[\text{End}, a, b, c] \wedge$
 $(b \neq a \wedge b \neq c \wedge a \neq c \wedge (b = b \vee (a = b \wedge c = b))) \wedge \text{End}(b) \supset$
 $\forall w . \text{End}(w) \supset w = b$

Simplify, by eliminating the atoms in the second main conjunct which are implied by the first:

$S[\text{End}, a, b, c] \wedge$
 $\text{End}(b) \supset \forall w . \text{End}(w) \supset w = b$

Now reason by cases. Suppose $\text{End}(b)$. Then

$\forall w . \text{End}(w) \supset w = b$

Since $c \neq b$, this means $\neg \text{End}(c)$. Together with $S[\text{End}, a, b, c]$, this yields $\text{End}(a)$. On the other hand, suppose $\neg \text{End}(b)$. Then again, it must be the case that $\text{End}(a)$.

So far we've shown that $\text{End}(a)$ is a consequence of the circumscription. Reinstantiating the schema strengthens the conclusion to that a is the only member of End .

Let $p = (\lambda u . u = a)$
 $x = a$
 $y = b$
 $z = c$

It is straightforward to show that the instantiated schema:

$S[\text{End}, a, b, c] \wedge$
 $(S[(\lambda u . u = a), a, b, c] \wedge \text{End}(a)) \supset \forall w . \text{End}(w) \supset w = a$

implies

$$\text{End}(a) \supset \forall w . \text{End}(w) \supset w=a$$

Thus, circumscribing End with a, b, and c varying forces End to have cardinality 1:

$$\text{Circum}(S[\text{End}, a, b, c], \text{End}) \vdash \text{End}(a) \wedge \forall w . \text{End}(w) \supset w=a$$

4.3. Example: The Cooking World, Continued

We continue with the example of plan recognition in the cooking world. There are two observations. The first is described in the previous chapter: the cook is reported to be making spaghetti or fettucini. We concluded that the cook was making some kind of pasta dish. The second observation is that the cook is making marinara sauce. From the second observation alone one could conclude that one of the dishes involving marinara sauce, namely spaghetti marinara or chicken marinara, was being prepared. If we allow the possibility that the two observations are unrelated, then all that one can conclude is the conjunction of the two conclusions; namely,

$$\begin{aligned} &\exists x . \text{MakePastaDish}(x) \wedge \\ &\exists y . (\text{MakeSpaghettiMarinara}(y) \vee \text{MakeChickenMarinara}(y)) \end{aligned}$$

But consider what holds in all the minimum covering models. In such a model, there is only one End events (in the present example). Each of the variables x and y must be interpreted as End event, since the types MakePastaDish, MakeSpaghettiMarinara, and MakeChickenMarinara all specialize End. Therefore x and y must be interpreted as the same entity. The disjointedness assumptions tell us that an entity cannot be both a MakePastaDish and a MakeChickenMarinara. Therefore the cook must be making spaghetti marinara in all minimum covering models. This is the conclusion mc-entailed by the observations.

Following is a proof sketch. The first part of the proof appears in the previous chapter.

Second Observation

$$\text{MakeMarinara}(M2)$$

Upward Inference, Existential Instantiation

$$\text{MakeSpaghettiMarinara}(*I2) \vee \text{MakeChickenMarinara}(*I2)$$

Abstraction

$\text{MakePastaDish}(*I2) \vee \text{MakeMeatDish}(*I2)$

Abstraction

$\text{EndEvent}(*I2)$

Conclusion from first Observation (Chapter 3)

$\text{EndEvent}(*I1)$

Strongest Minimality Assumption

$\forall x, y . \text{EndEvent}(x) \wedge \text{EndEvent}(y) \supset x=y$

Universal Instantiation & Modus Ponens

$*I1 = *I2$

Conclusion from first Observation (Chapter 3)

$\text{MakePastaDish}(*I1)$

Substitution of Equals

$\text{MakePastaDish}(*I2)$

Disjointedness Assumption

$\forall x . \neg \text{MakePastaDish}(x) \vee \neg \text{MakeMeatDish}(x)$

Disjunction Elimination

$\neg \text{MakeMeatDish}(*I2)$

Abstraction

$\text{MakeChickenMarinara}(*I2) \supset \text{MakeMeatDish}(*I2)$

Modus Tolens

$\neg \text{MakeChickenMarinara}(*I2)$

Disjunction Elimination

$\text{MakeSpaghettiMarinara}(*I2)$

Chapter 5

Incremental Recognition

A recognition problem is *incremental* when the recognizer is presented with a temporal *sequence* of observations, $\Gamma_1, \Gamma_2, \dots$. At any point in the sequence, the recognizer can be queried as to the consequences of the observations made so far.

Suppose mc-entailment is used to model the recognition process. Then after observation Γ_n , the recognizer can conclude Ω just in case

$$\Gamma_1 \cup \dots \cup \Gamma_n \vdash_{mc} \Omega$$

The mc-entailment relation is non-monotonic. As Γ_i are added to the left-hand side, the set of conclusions may no longer include Ω . Consider the following specific case.

Each observation is of the occurrence of a single event. Each observed event must be a component of an End event; call these End events $N_1, N_2, \dots, N_n$. Let there be a minimum covering model for $\Gamma_1, \dots, \Gamma_n$ in which the extension of End has cardinality 1; that is, $N_1 = N_2 = \dots = N_n$. If the number of observations is of reasonable size, it will quite often be the case that there is only a single specialization of End which could account for all of the observations. In particular, suppose that

$$\Gamma_1 \cup \dots \cup \Gamma_n \vdash_{mc} \exists x. E_k(x)$$

Now suppose the $n+1$ st observed event cannot be part of the same End event as all the previous ones. A minimum cover for $\Gamma_1, \dots, \Gamma_{n+1}$ must be of size 2. The recognizer can no longer assume that $N_1 = N_2 = \dots = N_n$. Instead, it may be the case that $N_1 = N_{n+1}$, and all the other N_j are equal; or that $N_2 = N_{n+1}$; and so on. There may be as many as 2^n different ways of grouping the observations, and each could correspond to a different minimum covering model. The conclusion $\exists x. E_k(x)$ must be withdrawn, and replaced with a long disjunction of different specializations of End.

5.1. Deficiencies in the MC Model

This kind of non-monotonic behavior is justified as a normative theory of what an agent should come to believe, if the minimization of End events is the only basis for combining information from different observations. Objections can be raised on the

grounds of complexity and psychological plausibility. The theory of mc-entailment can be adjusted, however, to both have a more efficient computational realization, and to more accurately reflex some of our intuitions about incremental recognition processes.

5.1.1. The Combinatorial Problem

Once the minimum cardinality default is weakened to allow two or more distinct End events, the number of ways of *grouping* the observations together as components of the same End event grows exponentially. In some domains the various constraints associated with event types could quickly rule out most of these possibilities. But it seems clear that additional principles will eventually be needed to deal with realistically-sized problems.

5.1.2. The Persistence Problem

Related to the combinatorial problem is the failure of mc-entailment to match up with our intuitions about the way a person would go about solving an incremental recognition problem. Once several pieces of information are "tied together", it seems unnatural to break that connection, simply because some seemingly *unrelated* piece of information comes along.

Consider the following (very!) short story, bearing in mind the event hierarchy in the beginning of Chapter 1.

Leo needed some cash. He took out his shotgun. Then he went to the bank. Later that day, he was seen in the woods.

As we read along, the descriptions of the Leo's getting a gun and going to the bank, together with his state of needing money, invoke the bank robbery event (or plan or script or ...). Walking in the woods is not part of robbing a bank. None the less, the story still seems to be about a bank robbery; we don't consider the possibility that Leo got his gun in order to go hunting in the woods. After the first three sentences, we conclude something like

$$\exists x . \text{RobBank}(x) \wedge \text{agent}(x)=\text{Leo}$$

and this belief *persists* even after the last sentence. Mc-entailment would justify this conclusion after the first three sentences, but after the last would only justify the weaker conclusion:

$\exists x, y .$

$(\text{RobBank}(x) \wedge (\text{Hunt}(y) \vee \text{Go-Hiking}(y))) \vee$
 $(\text{CashCheck}(y) \wedge \text{Hunt}(x))$

What is going on? One explanation is that we are minimizing End events in an incremental fashion. If we have to increase the number of End events to account for the story, we do, but we try not to *discard* any previous conclusions. Of course, new information could *force* us to withdraw a previous conclusion. So if the story continued

Leo had spent a long time in line at the bank, waiting to cash his paycheck. He was glad to finally be out stalking the wild moose.

the earlier conclusion is eliminated. But such stories appear either awkward or deliberately misleading (as a mystery story *should* be); making an analogy with syntax, one might call them *garden-path* stories.¹

This manner of reasoning is understandable if one views the times *between* observations as points at which the recognizer *accepts* his non-deductive conclusions as *full beliefs*. These beliefs, together with whatever else the recognizer believes, are subject to *minimal revision* as contradictory information is learned. But new *non-deductive* conclusions are not (in this model) the basis for revising old beliefs.

The difficulty with this model is that the minimum cardinality defaults are *always* revised when the number of End events must be increased. How then can conclusions *based* on these assumptions be retained? Section 5.3 below develops a model theory which exhibits this behavior, by separating *existential* and *universal* conclusions: the former may *persist* while the latter are rejected.

5.2. Refining the Model

This section considers a number of factors that could go into a theory of incremental recognition.

5.2.1. Incrementally Minimize Cardinality

¹ A garden-path sentence is a syntactically "correct" sentence which is difficult to understand, because the hearer makes incorrect decisions about its constituent structure before the end of the sentence is reached. The standard example is the sentence, "The horse raced past the barn fell."

Both the persistence problem and some aspects of the complexity problem are addressed by incrementally minimizing the number of End events, as described above. Most of the plan recognition systems described in the literature perform an incremental minimization, usually together with some of the following heuristics and constraints. Chapter 7 describes an algorithm for a plan recognizer which implements this heuristic.

5.2.2. Sticky Covers

A heuristic for grouping observations described in [Huff & Lesser 82] is to associate each new observation with the *most recent* observation with which it can consistently grouped. We call this the *sticky covering* assumption, since each observation tries to "stick" to the previous one. The sticky heuristic solves the combinatorial problem, as discussed in detail in Section 7.9.3, but is even more sensitive to garden-path type errors. An algorithm for calculating conclusions true in all sticky covers is given in Chapter 7, but we have not tried to work out a corresponding model theory.

5.2.3. Discourse Clues

Much work in computational models of discourse attempts to find clues which indicate how the various utterances which make up an extended discourse should be grouped. These take into consideration such features as "clue words" (such as "but" and "therefore") [Grosz 77] and intonation [Pierrehumbert 77].

5.2.4. Likelihood Associations

Eventually we'll need to deal with the frequency with which an event appears as a component of its different uses. The less common uses may be ignored if possible. Such a quantitative approach is not antithetical to the work described here. Once our qualitative methods choose all the possible interpretations of the data under some set of simplicity heuristics, probabilistic methods can be applied to rank and choose between the alternatives. This is exactly the approach taken by expert systems which employ notions similar to covering models, as described in Sections 1.4 and 6.4.

5.3. Model Theory for Incremental Minimization

The section defines a weaker version of minimum covering model, and then uses it to define an incremental operator, *imc-entailment*.

5.3.1. Minimum Covering Submodels

Mc-entailment justifies some kinds of conclusions that one would not ordinarily draw from a set of observations. Suppose that one observes A and B, and C is the only End act which entails both A and B. Then would seem reasonable to conclude C, and mc-entailment justifies this conclusion. However, mc-entailment also justifies the conclusion that no End act other than C occurs; and that assumption was the basis for the previous conclusion. Yet the statement that only one event will ever occur, at all times now and in the future, is almost surely false. The problem is that our intuitively acceptable conclusion C, has the same status as the dubious conclusion that nothing else will ever happen.

Simply eliminating the cardinality minimization would eliminate the doubtful conclusion, but at the cost of any combination of information from multiple observations. A more attractive approach is to weaken the notion of mc-entailment. Instead of minimizing the number of End events per se, we really only need to minimize the number of events which account for all the observations. There may be other End events which occur at other times and places, but they do not involve the current set of observed events. Instead of drawing conclusions based on the class of minimum covering models, one should draw conclusions based on the larger class of models which contain a minimum covering submodel.

Theorem 5.1 states that all the *existential* conclusions that one can obtain by considering minimum covering models can also be obtained by considering the larger class. This will provide the basis for retaining conclusions such as

$$\exists x . \text{RobBank}(x) \wedge \text{agent}(x)=\text{Leo}$$

while dropping the assumption that

$$\forall x,y . \text{End}(x) \wedge \text{End}(y) \supset x=y$$

5.3.1.1. Definitions

M_1 is a submodel of M_2 if the domain of M_1 is a subset of M_2 ; M_1 and M_2 agree on all predicates for tuples consisting only of elements in M_1 ; and M_1 and M_2 agree on the interpretation of all functions, restricted to the domain of M_1 . If M_1 is both a covering

model of Γ relative to H , and has a submodel M_2 which is a minimum cover for Γ relative to H , we say M_1 contains a minimum cover for Γ relative to H .

Ω is mcs-entailed by Γ , written

$$\Gamma_H \models_{\text{mcs}} \Omega$$

if Ω holds in all models which contain a minimum cover of Γ relative to H . For brevity, we will call a model which contains a minimum cover an *mcs-model*.

5.3.1.2. Theorem 5.1 (Non-Universal Conclusions)

Let Ω be a sentence which, when written with all quantifiers in initial position, contains no universal quantifiers. Then

$$\Gamma_H \models_{\text{mcs}} \Omega$$

if and only if

$$\Gamma_H \models_{\text{mc}} \Omega$$

5.3.2. Monotonic Incremental Recognition

The intuition behind a monotonic theory of incremental recognition is that after the recognizer obtains a piece of evidence, he believes forever the consequences of that evidence. This section defines the final operator which relates a sequence of observations to its *incrementally minimum covering entailed* conclusions. Theorem 5.2 states that this operator has the desired properties of acting like mc-entailment when there is only one observation, and allowing existential conclusions to persist as new observations are made.

5.3.2.1. Definitions

Let $\Gamma = (\Gamma_1, \Gamma_2, \dots, \Gamma_n)$ be an observation sequence. Following is a recursive definition of the class of incremental minimal covers of Γ . Let us say that M is a *n-candidate* if M is a covering model of Γ relative to H , and (if $n > 1$) M is an incremental minimum cover of $(\Gamma_1, \Gamma_2, \dots, \Gamma_n)$. Then M is an incremental minimal cover of Γ relative to H if M is an *n-candidate*, and:

(i) $n=1$, and M contains a minimum cover for Γ relative to H .

(ii) or $n > 1$, and M contains a submodel M_2 which has minimum cardinality in End among covering models of Γ which are submodels of n -candidates.

Ω is imc-entailed by $(\Gamma_1, \Gamma_2, \dots \Gamma_n)$, written

$$(\Gamma_1, \Gamma_2, \dots \Gamma_n) \vdash_H^{\text{imc}} \Omega$$

if Ω holds in all models which are incremental minimum covers of $(Q_1, Q_2, \dots Q_n)$ relative to H .

5.3.2.2. Theorem 5.2 (Incremental Recognition)

In the case of a single observation, imc-entailment is the same as mcs-entailment.

$$\begin{array}{l} (\Gamma_1) \vdash_H^{\text{imc}} \Omega \\ \text{if and only if} \\ \Gamma_1 \vdash_H^{\text{mcs}} \Omega \end{array}$$

In the case of multiple observations, imc-entailment is monotonic.

$$\begin{array}{l} (\Gamma_1, \dots \Gamma_{n-1}) \vdash_H^{\text{imc}} \Omega \\ \text{implies} \\ (\Gamma_1, \dots \Gamma_{n-1}, \Gamma_n) \vdash_H^{\text{imc}} \Omega \end{array}$$

Chapter 6

Examples

6.1. The Cooking World, Once More

The examples at the end of Chapters 3 and 4 illustrated inference from a disjunctive observation, and the combination of observations to uniquely identify the plan in progress. The next example illustrates two other important features of our system: the ability to handle disjunctive hypotheses, and to draw conclusions that reflect an abstract description of a class of possible plans.

Suppose that the observer learns that the cook is preparing some kind of sauce. This is not enough information to conclude a particular basic dish is being created: one cannot (as appears to be the case in most of the plan recognition systems described in Chapter 1) imagine that a *single* plan is evoked. Instead, one is justified in concluding that the cook is making some Pasta Dish, *or* Chicken Marinara. This disjunction collapses, via the abstraction axioms, to the fact that an event of type PrepareMeal is occurring. This final piece of information, non-specific as it is, may still be of great interest: for instance, if we are hungry! The example also illustrates the importance of adopting a formal framework that guarantees completeness: all possible uses of an event are considered. As noted in Chapter 3, the only *direct* use of MakeSauce is MakePastaDish. It is not clear whether any of the earlier heuristic-based plan recognition systems would consider MakeChickenMarinara, or if they would immediately (and unjustifiably) jump to the first conclusion. The proof so far is:

Observation

MakeSauce(Obs1)

Component/Use, Existential Instantiation

MakePastaDish(*I1) $\vee$ MakeChickenMarinara(*I1)

Abstraction

MakePastaDish(*I1) $\vee$ MakeMeatDish(*I1)

PrepareMeal(*I1)

End(*I1)

Now let the next observation be that the agent is making Noodles, and assume the End event inferred from the first observation is the same as the End event inferred from the second. Mc-entailment lets us conclude that a plan to make some Pasta Dish is in progress. We do not need (cannot, in fact) conclude that a *particular* kind of Pasta Dish is under construction.

Second Observation

MakeNoodles(Obs2)

Component Use, Existential Instantiation

MakePastaDish(*I2)

Abstraction

PrepareMeal(*I2)
End(*I2)

Strongest Minimality Assumption

$\forall x, y . \text{End}(x) \wedge \text{End}(y) \supset x=y$

Universal Instantiation & Modus Ponens

*I1 = *I2

Substitution of Equals

MakePastaDish(*I1)

Disjointedness Assumption

$\forall x . \neg \text{MakePastaDish}(x) \vee \neg \text{MakeMeatDish}(x)$

Disjunction Elimination

$\neg \text{MakeMeatDish}(*I1)$

Abstraction

$\text{MakeChickenMarinara}(*I1) \supset \text{MakeMeatDish}(*I1)$

Modus Tolens

$\neg \text{MakeChickenMarinara}(*I1)$

Disjunction Elimination

MakePastaDish(*I1)

Another example from this domain appears in Chapter 7 and in Appendix E (Transcripts). That example shows how one observation can constrain a different observation to be of specialized in a particular way. Specifically, a later observation of MakeMarinara constrains an earlier observation of MakeNoodles to be in fact an instance of MakeSpaghetti. Information can *flow* from one observation to another.

6.2. Indirect Speech Acts

A proper treatment of discourse requires careful attention to the representation of beliefs and intentions, as well as to the relation of an utterance to its meaning. An fully adequate treatment therefore necessitates the use of some sort of intensional or modal logic. None the less, we will try to approximate the problem of recognizing a speech act using the tools at hand, in order to suggest how the present approach might be extended to the discourse domain.

6.2.1. Representation

Recall that properties are represented by terms. The property that an agent A knows P is simply represented by applying the function know to terms for the agent and property P. The function can relates an agent and a property the agent can bring about. We assume that for every act (event) type, there is a property which holds just after an instance of the event occurs. Thus the fact that John knows at time T₁ that Mary can give John the salt might be written:

Holds(know(John,can(Mary,gave(Mary,John,salt))), T₁)

Finally we need the function knowif, which relate an agent and a property whose truth value the agent knows. Many objections can be raised to this notation, such as its referential transparency, and the omission of time indexes on the objects of beliefs, but it suffices for the example.

The following diagram shows part of the event hierarchy. The types SurfaceImperative and SurfaceQuestion classify utterances of commands and questions, respectively. Request and InformIf are speech act event type. A request can be specialized as a Command, a DirectRequest, or an IndirectRequest. Each of these specializations corresponds to different decomposition of the Request speech act, such as appears in [Litman 84]. Some of the discourse plans which employ speech acts are ObtainByAsking and FindOutByAsking. These discourse plans specialize more general methods for obtaining objects and finding out information. There may be a great deal

of structure above, or the discourse plans themselves may be taken as End events, and thus terminate analysis. The example here does not depend on any higher structure in the library.

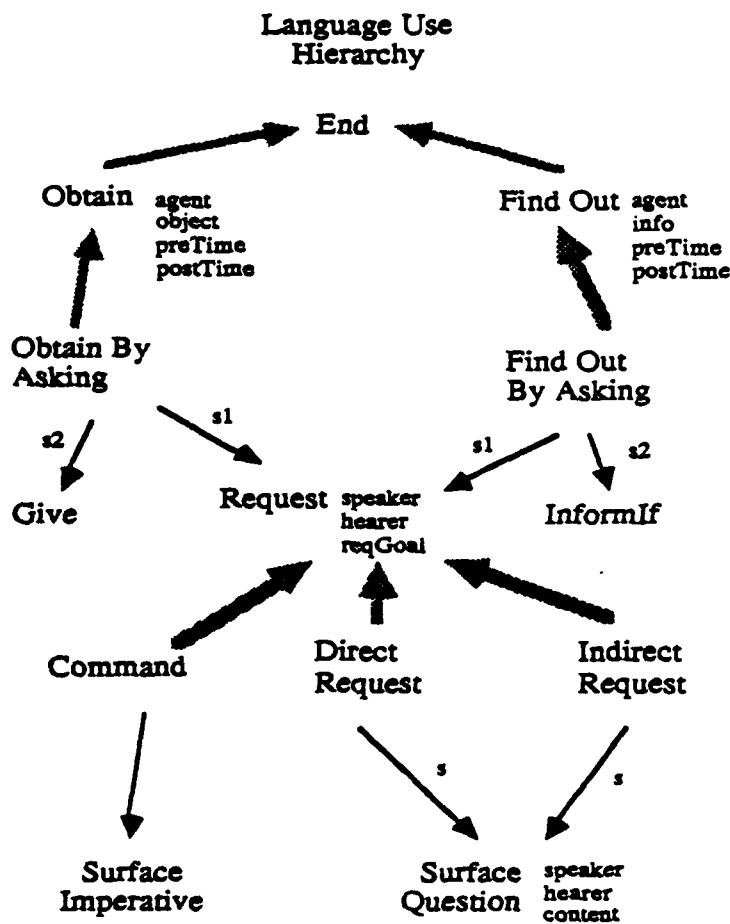


figure 6.1: Language Use Hierarchy

Following is an abbreviated list of the axioms for this hierarchy. All the actions have roles of agent and time, and other roles as specified.

- *Non-linguistic Acts:* Obtain has the role of the object to be obtained. During pre-Time the agent does not have the object; during postTime the agent does. Find-Out has the role info, a property, whose truth-value is to be determined. During pre-Time the agent does not knowif the info is true or false; during postTime the agent does.

$\forall x . \text{Obtain}(x) \supset$

Never(have(agent(x),obj(x)), preTime(x)) $\wedge$
 meets(preTime(x), time(x)) $\wedge$...
 Holds(have(agent(x),obj(x)), postTime(x)) $\wedge$
 meets(time(x), postTime(x)) $\wedge$...

$\forall x . \text{FindOut}(x) \supset$

Never(knowif(agent(x), info(x)), preTime(x)) $\wedge$
 meets(preTime(x), time(x)) $\wedge$
 Holds(knowif(agent(x), info(x)), postTime(x)) $\wedge$
 meets(time(x), postTime(x)) $\wedge$...

• *Discourse Acts:* The decomposition of ObtainByAsking states that the agent performs a Request, whose reqGoal is that the hearer gave the agent the object. This step, s1, is followed by step s2, in which the hearer gives the agent the object.

$\forall x . \text{ObtainByAsking}(x) \supset$

Request(s1(x)) $\wedge$
 reqGoal(s1(x)) = gave(hearer(s1(x)), agent(x), obj(x)) $\wedge$
 Give(s2(x)) $\wedge$...

FindOutByAsking is very similar; the decomposition states that the agent performs a Request, whose reqGoal is that the hearer has informed the agent of the truth value of the info. In s2 the affected performs an InformIf to the agent.

$\forall x . \text{FindOutByAsking}(x) \supset$

Request(s1(x)) $\wedge$
 reqGoal(s1(x)) = informedIf(hearer(s1(x)), agent(x), info(x)) $\wedge$
 InformIf(s2(x)) $\wedge$...

• *Speech Acts:* Each of the specializations of a Request contains a single *utterance act* in its decomposition. The single step in a DirectRequest is a surface yes/no question. The reqGoal of the DirectRequest must be that the affected has informed the agent of the truth value of the (propositional) content of the SurfaceQuestion.

$$\begin{aligned}\forall x . \text{DirectRequest}(x) \supset \\ \text{SurfaceQuestion}(s(x)) \wedge \\ \text{reqGoal}(x) = \text{informedIf}(\text{hearer}(x), \text{speaker}(x), \text{content}(s(x))) \wedge \dots\end{aligned}$$

The single step of an IndirectRequest is also a SurfaceQuestion; however, there is a different relation between the content of the question and the reqGoal of the request. The question must be a "can" question, of the form, "can the affected bring about the goal of the request?"

$$\begin{aligned}\forall x . \text{IndirectRequest}(x) \supset \\ \text{SurfaceQuestion}(s(x)) \wedge \\ \text{content}(s(x)) = \text{can}(\text{hearer}(x), \text{reqGoal}(x)) \wedge \dots\end{aligned}$$

6.2.2. Assumptions

The following component/use axioms are generated by assuming that the hierarchy is complete. Whenever a SurfaceQuestion occurs, it must be part of a DirectRequest or an IndirectRequest; and whenever a Request occurs, it must be part of an ObtainByAsking or FindOutByAsking.

$$\begin{aligned}\forall x . \text{SurfaceQuestion}(x) \supset \\ (\exists y . \text{DirectRequest}(y) \wedge x=s(y)) \vee \\ (\exists y . \text{IndirectRequest}(y) \wedge x=s(y)) \\ \forall x . \text{Request}(x) \supset \\ (\exists y . \text{ObtainByAsking}(y) \wedge x=s1(y)) \vee \\ (\exists y . \text{FindOutByAsking}(y) \wedge x=s1(y))\end{aligned}$$

6.2.3. The Problem

Suppose S says to H, "Can you give me the salt?" Real world knowledge (part of H_G) includes the statement that at all times, S knows whether or not H can give S the salt:

$$\forall t . \text{Hold}(\text{knowIf}(S, \text{can}(H, \text{gave}(H, S, \text{salt}))), t)$$

An instance of a SurfaceQuestion, Q1, occurs, with the content, "H can give S the salt".

$$\text{SurfaceQuestion}(Q1) \wedge \text{speaker}(Q1) = S \wedge \text{hearer}(Q1) = H \wedge \\ \text{content}(Q1) = \text{can}(H, \text{gave}(H, S, \text{salt}))$$

Now apply the upward inference assumption for SurfaceQuestion. It must be case that Q1 is either a step of some DirectRequest (call it *R1) or some IndirectRequest. In either case the constraints can be used to determine the possible reqGoals. In the direct case, the goal must be for H to tell S whether or not H can give S the salt. In the indirect case, the goal must be for H to give S the salt.

$$\text{DirectRequest}(*R1) \wedge Q1 = s(*R1) \wedge \\ \text{reqGoal}(*R1) = \text{informedIf}(H, S, \text{can}(H, \text{gave}(H, S, \text{salt})))$$

∨

$$\text{IndirectRequest}(*R1) \wedge Q1 = s(*R1) \wedge \\ \text{reqGoal}(*R1) = \text{gave}(H, S, \text{salt})$$

Neither alternative can (yet) be eliminated on the basis of inconsistent constraints. So we apply the second upward closure assumptions to this statement, yielding a four-way disjunction describing the act *R2 which has step *R1. Constraint information can now be used to eliminate all but one alternative.

FindOut Action

Direct Request : Precondition Fails

*R1 is a direct request, and the goal of *R1 is to be informed if H can pass the salt. *R1 is the first step of the FindOut action. However, the constraint that S does not know if H can pass the salt before the action occurs is provably false.

Indirect Request : Ill-Formed

*R1 is an indirect request, and the goal of *R1 is for H to pass the salt. *R1 is the first step of the FindOut action. However, to be part of FindOut, the goal of *R1 must be of the form informedIf(-), instead of gave(-). Therefore this alternative is ill-formed.

Obtain Action

Direct Request : Ill-Formed

*R1 is a direct request, and the goal of *R1 is to be informed if H can pass the salt. *R1 is the first step of the Obtain action. However, to be part of Obtain, the goal of *R1 must be of the form gave(-), instead of informedIf(-). Therefore this alternative is ill-formed.

Indirect Request : Accepted

*R1 is an indirect request, and the goal of *R1 is for H to pass the salt. *R1 is the first step of the Obtain action. This alternative is well formed, and no violated constraints can be found.

The follow statement encodes this analysis. An ✕ appears at the point in each alternative where an inconsistency is detected. The ill-formed alternatives are ruled out by the property identity axioms discussed in Section 2.2.

FindOutByAsking(*R2) ∧ *R1 = s1(*R2) ∧
DirectRequest(*R1) ∧ Q1 = s(*R1) ∧
reqGoal(*R1) = informedIf(H,S, can(H, gave(H,S,salt))) ∧
reqGoal(s1(*R2)) = informedIf(H,S, info(*R2)) ∧
info(*R2) = can(H, gave(H,S,salt)) ∧
✕ Never(knowif(S, can(H, gave(H,S,salt))), preTime(*R4))
∨
FindOutByAsking(*R2) ∧ *R1 = s1(*R2) ∧
IndirectRequest(*R1) ∧ Q1 = s(*R1) ∧
reqGoal(*R1) = gave(H,S,salt)
✕ reqGoal(s1(*R2)) = informedIf(H,S, info(*R2)) ∧
∨

$$\begin{array}{l}
\text{ObtainByAsking}(*R2) \wedge *R1 = s1(*R2) \wedge \\
\text{DirectRequest}(*R1) \wedge Q1 = s(*R1) \wedge \\
\text{reqGoal}(*R1) = \text{informedIf}(H,S, \text{can}(H, \text{gave}(H,S,\text{salt}))) \wedge \\
\times \quad \text{reqGoal}(s1(*R2)) = \text{gave}(H,S, \text{obj}(*R2)) \wedge \\
\vee \\
\text{ObtainByAsking}(*R2) \wedge *R1 = s1(*R2) \wedge \\
\text{IndirectRequest}(*R1) \wedge Q1 = s(*R1) \wedge \\
\text{reqGoal}(*R1) = \text{gave}(H,S,\text{salt}) \wedge \\
\text{reqGoal}(s1(*R2)) = \text{gave}(H,S,\text{object}(*R2)) \wedge \\
\checkmark \quad \text{object}(*R2) = \text{salt}
\end{array}$$

The final disjunct must be true: The recognizer is justified in concluding that S is performing the plan to obtain the salt by asking for it.

$$\begin{array}{l}
\text{ObtainByAsking}(*R2) \wedge \text{object}(*R2) = \text{salt} \wedge *R1 = s1(*R2) \wedge \\
\text{IndirectRequest}(*R1) \wedge \text{reqGoal}(*R1) = \text{gave}(H,S,\text{salt}) \wedge Q1 = s(*R1) \wedge \\
\text{SurfaceQuestion}(Q1) \wedge \text{content}(Q1) = \text{can}(H, \text{gave}(H,S,\text{salt}))
\end{array}$$

6.3. Operating Systems: Multiple Events

Several research groups have examined the use of plan recognition in "smart" operating systems, which could answer user questions and/or watch what the user was doing, and make suggestions about potential pitfalls and more efficient ways of accomplishing the same tasks. A user often works on several different tasks during a single session at the terminal, and frequently jumps back and forth between uncompleted tasks. Therefore a plan recognition system for this domain must be able to handle multiple concurrent unrelated plans. The very generality of the present approach is an advantage in this domain, where the focus-type heuristics used by other plan recognition systems are not so applicable.

6.3.1. Representation

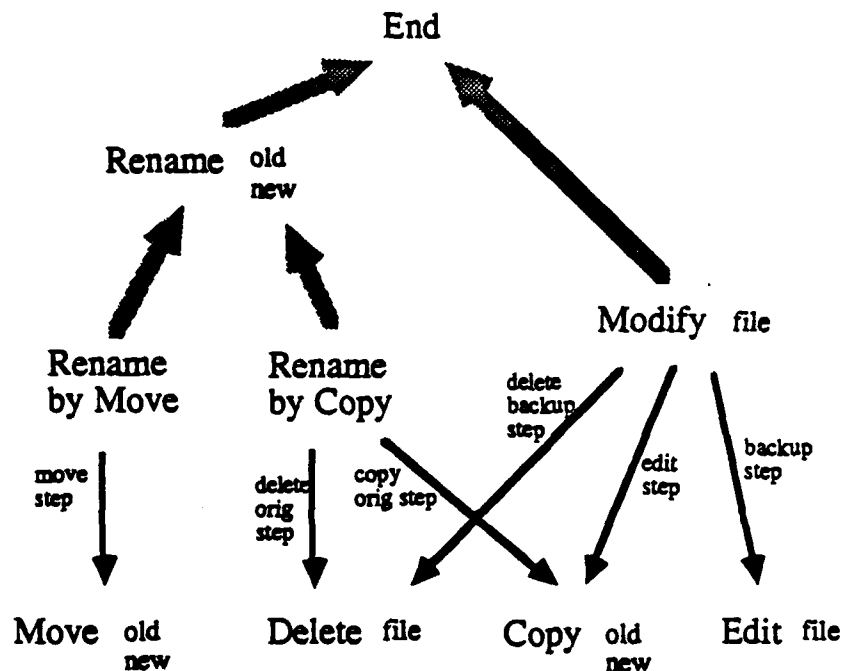


figure 6.2: Operating System Hierarchy

Consider the following hierarchy. There are two End plans: to Rename a file, and to Modify a file.

$$\forall x . \text{Rename}(x) \supset \text{End}(x)$$

$$\forall x . \text{Modify}(x) \supset \text{End}(x)$$

There are two ways to specialize the Rename event. A RenameByCopy involves Copying a file, and then Deleting the original version of the file, without making any changes in the original file.

$$\forall x . \text{RenameByCopy}(x) \supset \text{Rename}(x)$$

$$\begin{aligned}
\forall x . \text{RenameByCopy}(x) \supset & \\
& \text{Copy}(s1(x)) \wedge \\
& \text{Delete}(s2(x)) \wedge \\
& \text{old}(s1(x)) = \text{old}(x) \wedge \\
& \text{new}(s1(x)) = \text{new}(x) \wedge \\
& \text{file}(s2(x)) = \text{old}(x) \wedge \\
& \text{BeforeMeet}(\text{time}(s1(x)), \text{time}(s2(x))) \wedge \\
& \text{Never}(\text{modified}(\text{old}(x)), \text{time}(x)) \wedge \\
& \text{Starts}(\text{time}(s1(x)), \text{time}(x)) \wedge \\
& \text{Finishes}(\text{time}(s2(x)), \text{time}(x))
\end{aligned}$$

A better way to rename a file is to **RenameByMove**, which simply uses the **Move** command. A helpful system might suggest that a user try the **Move** command if it recognizes many instances of **RenameByCopy**.

$$\forall x . \text{RenameByMove}(x) \supset \text{Rename}(x)$$

$$\begin{aligned}
\forall x . \text{RenameByMove}(x) \supset & \\
& \text{Move}(s1(x)) \wedge \\
& \text{old}(s1(x)) = \text{old}(x) \wedge \\
& \text{new}(s1(x)) = \text{new}(x)
\end{aligned}$$

The **Modify** command has three steps. In the first, the original file is backed up by **Copying**. Then the original file is **Edited**. Finally, the backup copy is deleted.

$$\begin{aligned}
\forall x . \text{Modify}(x) \supset & \\
& \text{Copy}(s1(x)) \wedge \\
& \text{Edit}(s2(x)) \wedge \\
& \text{Delete}(s3(x)) \wedge \\
& \text{old}(s1(x)) = \text{file}(x) \wedge \\
& \text{file}(s2(x)) = \text{file}(x) \wedge \\
& \text{file}(s3(x)) = \text{new}(s1(x)) \wedge \\
& \text{BeforeMeet}(\text{time}(s1(x)), \text{time}(s2(x))) \wedge \\
& \text{BeforeMeet}(\text{time}(s2(x)), \text{time}(s3(x)))
\end{aligned}$$

6.3.2. Assumptions

Following are some of the statements obtained by minimizing the hierarchy. The Component/Use assumptions include the statement that every Copy action is either part of a RenameByCopy or of a Modify.

$$\begin{aligned} \forall x . \text{Copy}(x) \supset \\ (\exists y . \text{RenameByCopy}(y) \wedge x=s1(y)) \vee \\ (\exists y . \text{Modify}(y) \wedge x=s1(y)) \end{aligned}$$

Every Delete event is either the second step of a RenameByCopy, or the third step of a Modify, in any covering model.

$$\begin{aligned} \forall x . \text{Delete}(x) \supset \\ (\exists y . \text{RenameByCopy}(y) \wedge x=s2(y)) \vee \\ (\exists y . \text{Modify}(y) \wedge x=s3(y)) \end{aligned}$$

6.3.3. The Problem

Suppose the plan recognition system observes each action the user performs. Whenever a new file name is typed, the system generates a constant with the same name, and asserts that that constant is not equal to any other file name constant. (We don't allow UNIX™ style "links"!) During a session the user types the following commands.

- (1) % copy foo bar
- (2) % copy jack sprat
- (3) % delete foo

The system should recognize two concurrent plans. The first is to rename the file "foo" to "bar". The second is to either rename or modify the file "jack". Let's examine how these inferences could be made.
Statement (1) is encoded:

$$\text{Copy}(C1) \wedge \text{old}(C1)=\text{foo} \wedge \text{new}(C1)=\text{bar}$$

The decomposition completeness axiom for Copy lets the system infer that C1 is either part of a RenameByCopy or Modify. A new name *I1 is generated (by existential instantiation) for the disjunctively-described event.

$$\begin{aligned} & \text{End}(*I1) \wedge \\ & (\quad \text{RenameByCopy}(*I1) \wedge C1=s1(*I1)) \\ & \quad \vee \\ & \quad \text{Modify}(*I1) \wedge C1=s1(*I1)) \\ &) \end{aligned}$$

Statement (2) is encoded:

$$\text{Copy}(C2) \wedge \text{old}(C2)=\text{jack} \wedge \text{new}(C2)=\text{spratt} \wedge \text{Before}(\text{time}(C1), \text{time}(C2))$$

Again the system creates a disjunctive description for the event *I2, which has C2 as a component.

$$\begin{aligned} & \text{End}(*I2) \wedge \\ & (\quad \text{RenameByCopy}(*I2) \wedge C2=s1(*I2)) \\ & \quad \vee \\ & \quad \text{Modify}(*I2) \wedge C2=s1(*I2)) \\ &) \end{aligned}$$

The next step is to minimize the number of End events. The system might attempt to apply the strongest minimization default, that

$$\forall x,y . \text{End}(x) \wedge \text{End}(y) \supset x=y$$

However, doing so would lead to a contradiction. Because the types RenameByCopy and Modify are disjoint, *I1=*I2 would imply that C1=C2; however, the system knows that C1 and C2 are distinct -- among other reasons, their times are known to be not equal. The next strongest minimality default, that there are two End events, cannot lead to any new conclusions.

Statement (3), the act of deleting "foo", is encoded:

$$\text{Delete}(C3) \wedge \text{file}(C3)=\text{foo} \wedge \text{Before}(\text{time}(C2), \text{time}(C3))$$

The system infers by the decomposition completeness assumption for Delete that the user is performing a RenameByCopy or a Modify. The name *I3 is assigned to the inferred event.

$$\begin{aligned} & \text{End}(*I3) \wedge \\ & (\quad \text{RenameByCopy}(*I3) \wedge C3=s2(*I3)) \\ & \quad \vee \\ & \quad \text{Modify}(*I3) \wedge C3=s3(*I3)) \\ &) \end{aligned}$$

Again the system tries to minimize the number of End events. The second strongest minimality default says that there are no more than two End events.

$$\begin{aligned} \forall x,y,z . \text{End}(x) \wedge \text{End}(y) \wedge \text{End}(z) \supset \\ x=y \vee x=z \vee y=z \end{aligned}$$

In this case the formula is instantiated as follows.

$$*I1=*I2 \vee *I1=*I3 \vee *I2=*I3$$

We've already explained why the first alternative is impossible. Thus the system knows

$$*I1=*I3 \vee *I2=*I3$$

The system then reduces this disjunction by reasoning by cases. Suppose that $*I2=*I3$. This would mean that the sequence

(2) % copy jack sprat

(3) % delete foo

is either part of a RenameByCopy or of a Modify, described as follows.

$$\begin{aligned} & \text{End}(*I2) \wedge \\ & (\quad \text{RenameByCopy}(*I2) \wedge C2=s1(*I2) \wedge C3=s2(*I2) \wedge \\ & \quad \times \quad \text{old}(*I2) = \text{jack} \wedge \text{old}(*I2) = \text{foo}) \\ & \quad \vee \\ & \quad \text{Modify}(*I2) \wedge C2=s1(*I2) \wedge C3=s3(*I2) \wedge \\ & \quad \text{file}(s3(*I2)) = \text{foo} \wedge \\ & \quad \text{new}(s1(*I2)) = \text{sprat} \wedge \\ & \quad \times \quad \text{file}(s3(*I2)) = \text{new}(s1(*I2)))) \\ &) \end{aligned}$$

But both disjuncts are impossible, since the files which appear as roles of each event do not match up (as marked with X's). Therefore, if the minimality default holds, it must be the case that

$$*I1 = *I3$$

This means that the observations

(1) % copy foo bar

(3) % delete foo

should be grouped together, as part of a Rename or Modify. This assumption leads the system to conclude the disjunction:

```

End(*I1) ^
(
  (RenameByCopy(*I1) ^ C1=s1(*I1) ^ C3=s2(*I1) ^
    old(*I1) = foo ^ new(*I1) = bar )
  v
  (Modify(*I1) ^ C1=s1(*I1) ^ C3=s3(*I1) ^
    file(s3(*I1)) = foo ^
    new(s1(*I1)) = bar ^
    X file(s3(*I1)) = new(s1(*I1))) )
)

```

The second alternative is ruled out, since the actions cannot be part of the same Modify. The system concludes observations (1) and (3) make up a RenameByCopy act, and observation (2) is part of some unrelated End action.

```

End(*I1) ^
RenameByCopy(*I1) ^
old(*I1) = foo ^ new(*I1) = bar ^
End(*I2) ^
(
  (RenameByCopy(*I2) ^ C2=s1(*I2) )
  v
  (Modify(*I2) ^ C2=s1(*I2) )
)

```

Further checking of constraints may be performed, but no inconsistency will arise. At this point the plan recognizer may trigger the "advice giver" to tell the user:

```
*** You can rename a file by typing
*** % move oldname newname
```

6.4. Medical Diagnosis

As discussed in Section 1.4, there are close links between the kinds of reasoning involved in plan recognition, and that employed in medical diagnosis. The following example is drawn from [Pople 82], and was handled by CADUCEUS, an expert system that deals with internal medicine. We have made a number of simplifications, but the key points of the example remain. These include the need to consider specializations of an pathological state (abstract event type) in order to explain a symptom, finding, or state, and the process of combining or unifying the tasks invoked by each finding. This combination process corresponds to the minimization of End events in the plan recognition framework.

6.4.1. Representation

The figure below illustrates a small part of CADUCEUS's knowledge base. The thin "component" arcs are here understood as meaning "can cause". We've added the type End as an abstraction of all pathological states which are not caused by other pathological states. The *basic* specializations of End are called *specific disease entities*. We've simplified the hierarchy by making the specializations of anemia and shock specific disease entities; in the actual knowledge base, anemia and shock are caused by other conditions.

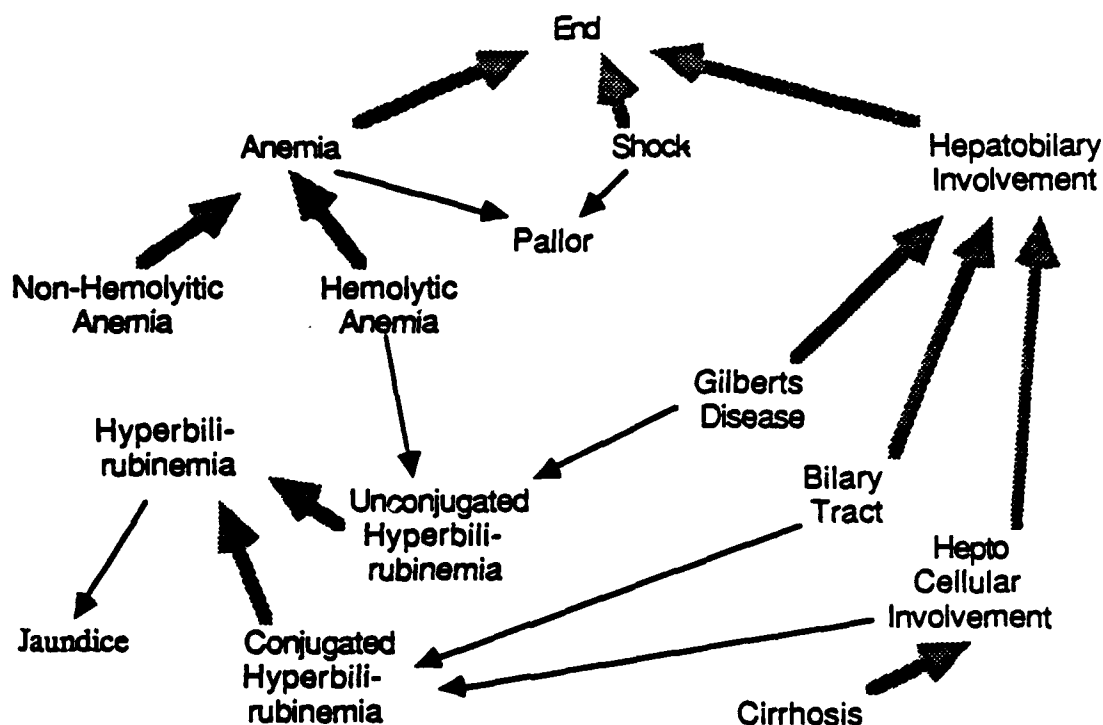


figure 6.3: Medical Hierarchy

The logical encoding of this network is as expected. The symptoms caused by a disease appear in the decomposition axiom for that disease. This is a considerable simplification over the original CADUCEUS model, in which the causal connections need only be *probable*. Symptoms of high probability, however, are taken by CADUCEUS as *necessary* manifestations, and CADUCEUS will *rule out* a disease on the basis of the *absence* of such symptoms. The *constraints* that appear at the end of a decomposition axiom would include conditions of the patient that are (very nearly) necessary for the occurrence of the disease, but are not themselves pathological. These could include the age and weight of the patient, his immunization history, and so on. Thus a constraint on Alzheimer's disease would include the fact that the patient is over 40.

A few of the axioms and assumptions follow. All kinds of hyperbilirubinemia cause Jaundice, and both anemia and shock cause pallor.

$$\forall y . \text{hyperbilirubinemia}(y) \supset \text{jaundice}(j(y))$$

$$\forall y . \text{anemia}(y) \supset \text{pallor}(p(y))$$

$$\forall y . \text{shock}(y) \supset \text{pallor}(\text{p}(y))$$

Unconjugated hyperbilirubinemia is a kind of hyperbilirubinemia, which can be caused by hemolytic anemia, a kind of anemia.

$$\forall x . \text{unconjugated-hyperbilirubinemia}(x) \supset \text{hyperbilirubinemia}(x)$$

$$\forall y . \text{hemolytic-anemia}(y) \supset \text{hyperbilirubinemia}(\text{h}(y))$$

$$\forall x . \text{hemolytic-anemia}(x) \supset \text{anemia}(x)$$

It may seem a bit odd that we need to use a first-order language, when the problem would seem to be expressible in purely propositional terms. The problem with using propositional logic arises from the abstract pathological states. A realistic medical knowledge base incorporates several methods of classifying diseases, leading to a complex and intertwined abstraction hierarchy. It is very likely that any patient will manifest at least two distinct pathological states (perhaps causally related) that specialize the same state. In a purely propositional system such a pair would appear to be competitors, as in a differential diagnosis set. But this would plainly be incorrect, if the two states were causally related.

6.4.2. Assumptions

Now we restrict our attention to the covering models of this hierarchy. The *exhaustiveness assumptions* include the fact that every case of hyperbilirubinemia is either conjugated or unconjugated.

$$\begin{aligned} \forall x . \text{unconjugated-hyperbilirubinemia}(x) \supset \\ \text{conjugated-hyperbilirubinemia}(x) \vee \\ \text{unconjugated-hyperbilirubinemia}(x) \end{aligned}$$

Disjointedness assumptions include the fact that the pathological states of anemia, shock, and hepatobiliary involvement are distinct. It is important to note that this does *not* mean that the states cannot occur simultaneously; rather, that none of these states abstract each other.

$$\forall x . \neg \text{anemia}(x) \vee \neg \text{shock}(x)$$

$$\forall x . \neg \text{anemia}(x) \vee \neg \text{hepatobiliary-involvement}(x)$$

$$\forall x . \neg \text{shock}(x) \vee \neg \text{hepatobiliary-involvement}(x)$$

Finally, the *component/use assumptions*, better called the *manifestation/cause assumptions*, allow one to conclude the disjunction of causes of a pathological state, thus creating a *differential diagnosis* set. An important special case occurs when there is only one cause, usually at a fairly high level of abstraction, for a state. An example of this is the association of jaundice with hyperbilirubinemia. (Pople calls this case a *constrictor relationship* between the manifestation and cause, and argues that such cases play a critical role in reducing search in diagnostic problem solving.) A less conclusive assumption says that pallor indicates anemia or shock.

$$\forall x . \text{jaundice}(x) \supset \exists y . \text{hyperbilirubinemia}(y) \wedge x=j(y)$$

$$\begin{aligned} \forall x . \text{pallor}(x) \supset \\ (\exists y . \text{anemia}(y) \wedge x=p(y)) \vee \\ (\exists y . \text{shock}(y) \wedge x=p(y)) \end{aligned}$$

6.4.3. The Problem

We'll sketch the kind of reasoning that goes on when the diagnostician is confronted with two symptoms, jaundice and pallor. From jaundice the diagnostician concludes hyperbilirubinemia. This leads (by exhaustion) to either the conjugated or unconjugated varieties. Now CADUCEUS (and perhaps a human physician?) may try to perform tests to decide between these alternatives at this point. The framework we've developed, however, allows us continue inference in each alternative. The first leads to hemolytic anemia and then anemia; the second to three different kinds of hepatobiliary involvement. The following graph shows the final conclusion.

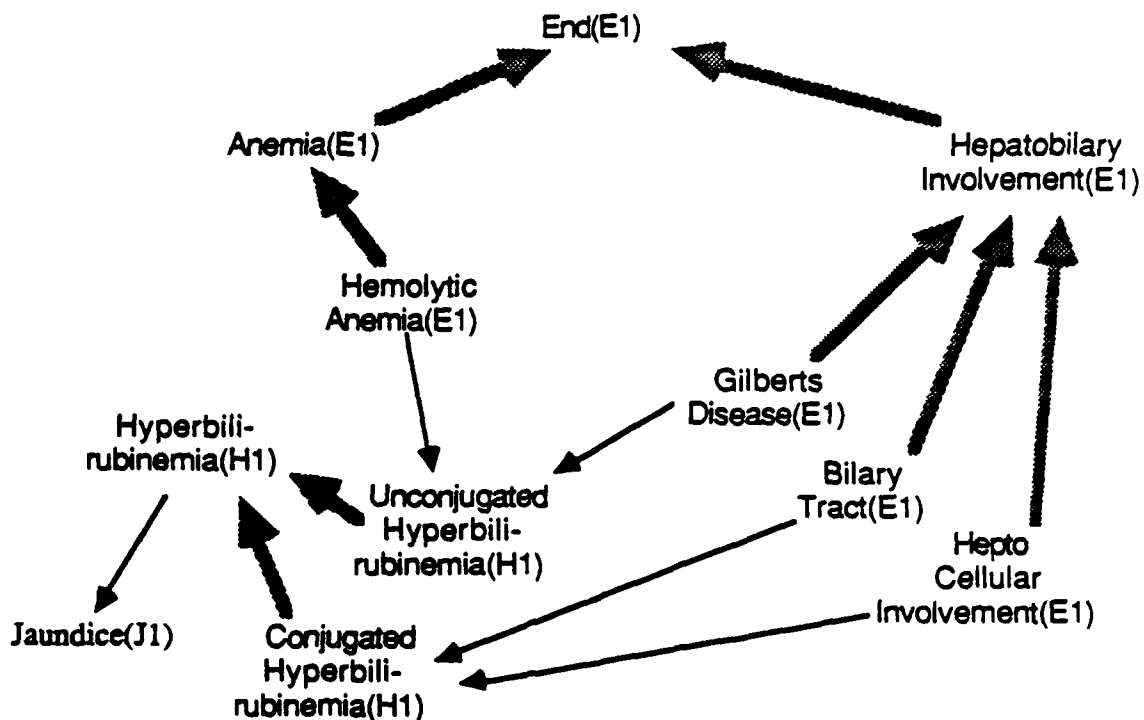


figure 6.4: Conclusions from Jaundice

The graph represents the following logical sentence. (Further steps in this example will be illustrated only in the graphical form.)

$$\begin{aligned}
 & \text{jaundice}(*J1) \wedge \text{hyperbilirubinemia}(*H1) \wedge \\
 & (\quad (\quad \text{unconjugated-hyperbilirubinemia}(*H1) \wedge \\
 & \quad \quad \text{hemolytic-anemia}(*E1) \wedge \\
 & \quad \quad \text{anemia}(*E1) \\
 & \quad) \\
 & \vee \\
 & (\quad \text{conjugated-hyperbilirubinemia}(*H1) \wedge \\
 & \quad (\quad \text{Gilberts-disease}(*E1) \vee \\
 & \quad \quad \text{biliary-tract}(*E1) \vee \\
 & \quad \quad \text{heptocellular-involvement}(*E1) \\
 & \quad) \wedge \\
 & \quad \text{hepatobiliary-involvement}(*E1) \\
 &) \\
 &) \wedge \\
 & \text{End}(*E1)
 \end{aligned}$$

Next the diagnostician considers pallor. This leads to a simple disjunction.

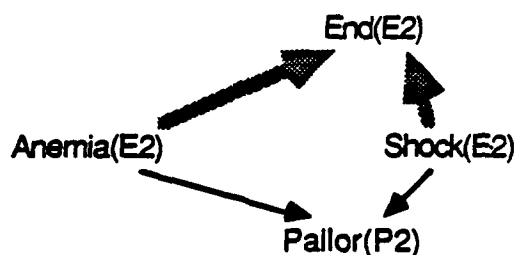


figure 6.5: Conclusions from Pallor

Finally, the diagnostician applies Occam's razor, by making the assumption that the symptoms are caused by the same disease. This corresponds to equating the specific disease entities at the highest level of abstraction (End) in each of the previous conclusions. In other words, we apply the strongest minimum cardinality default. This allows the diagnostician to conclude that the patient is suffering from hemolytic anemia, which has led to unconjugated hyperbilirubinemia.

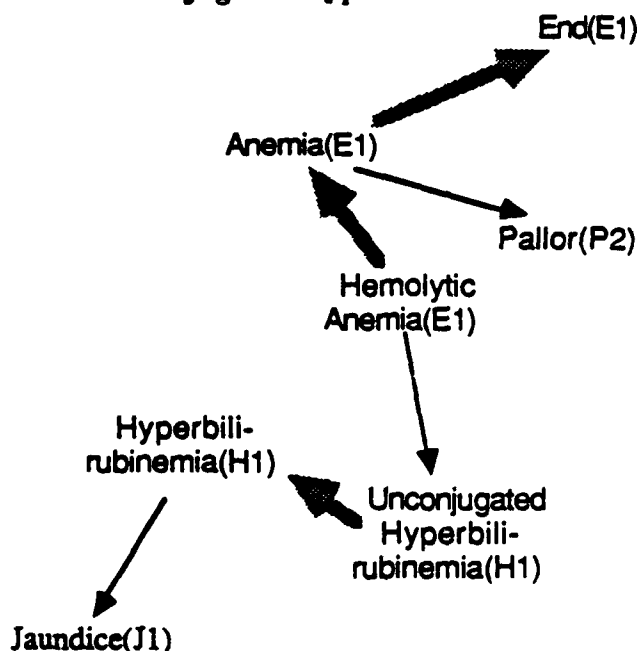


figure 6.6: Conclusions from Jaundice and Pallor

A practical medical expert system must deal with a great many problems not illustrated by this simple example. We have not dealt with the whole problem of generating appropriate tests for information; we cannot assume that the expert system is pas-

sively soaking in information, like a person reading a book. The full CADUCEUS knowledge base is enormous, and it is not clear whether the complete algorithms discussed in the next chapter could deal with it. It does seem plain, however, that our framework for plan recognition does formalize some key parts of diagnostic reasoning. The rich and highly structured knowledge base required for medical diagnosis makes it a very good domain for exercising any formal theory of non-deductive inference.

Chapter 7

Algorithms for Plan Recognition

7.1. Directing Inference

Two related problems arise in implementing our theory of plan recognition: inference must be *directed* toward some particular goal, and must be *limited* in some manner to insure that our programs don't run forever. The pattern of inference apparent in the previous examples suggests some answers: from each observation, apply Component/Use assumptions until an instance of type End is reached. That is, create a *proof* that some instance of End occurs. Reduce the number of alternatives (or *cases* in the proof) by checking constraints *locally*. In order to combine information from two observations, *equate* the instances of End inferred from each and propagate the equality, further reducing disjunctions. If all alternatives are eliminated, then conclude that the observations belong to distinct End events. Multiple simultaneous End events can be recognized by considering all ways of *grouping* the observations.

These operations suggest a *graph* based implementation, rather than one which stores sentences in clausal form. A graph is built bottom-up, from a node which represents an observed event, to a node which represents an End event. When we need to infer up the *abstraction* hierarchy, we create a new node of the more abstract kind, and record the the more specialized node as an *alternative for* the abstract node. If this is done properly, the result can be viewed as an and/or graph, *rooted at the End node*. The alternative (OR) arcs are marked with an equality sign ($=$), and the component (AND) arcs are labeled with the role function name. The figure below shows the graph that is generated an observation of MakeMarinara.

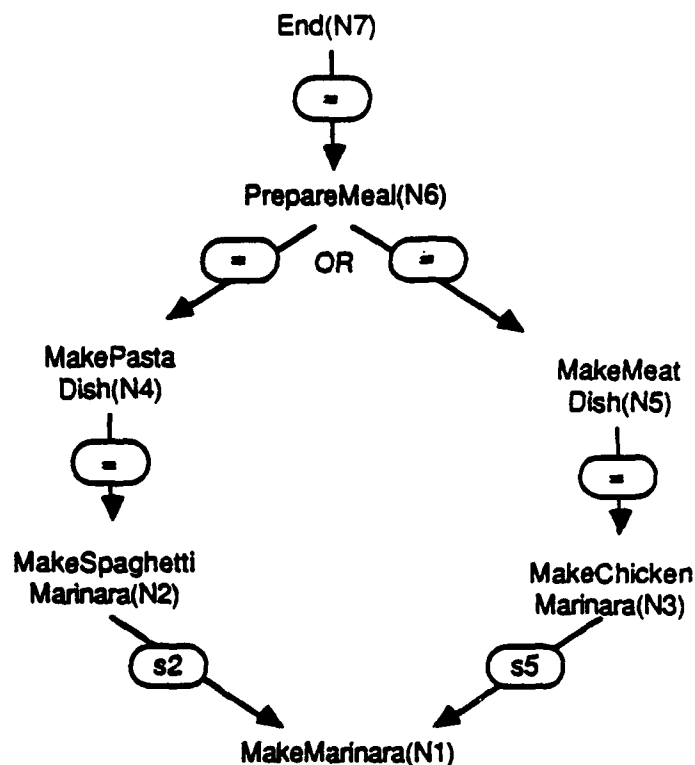


figure 7.1: E-Graph for MakeMarinara

These structures are called *explanation graphs*, or *e-graphs*. The graph has *two* complementary interpretations. It can be viewed as a *proof* that an instance of End occurs, given instances of the events in its leaves. It can also be interpreted as a *sentence* of FOL, which states that an instance of End occurs, that this instance is equal to one of a number of other tokens of more specific type, and so on. The graph above can be interpreted as the sentence

$$\begin{aligned}
 &\exists n_7, n_6, n_5, n_4, n_3, n_2, n_1 . \text{End}(n_7) \wedge n_7 = n_6 \wedge \text{PrepareMeal}(n_6) \wedge \\
 &\quad ((n_6 = n_4 \wedge \text{MakePastaDish}(n_4) \wedge \\
 &\quad \quad n_4 = n_2 \wedge \text{MakeSpaghettiMarinara}(n_2) \wedge \\
 &\quad \quad \text{step2}(n_2) = n_1 \wedge \text{MakeMarinara}(n_1))) \\
 &\quad \vee (n_6 = n_5 \wedge \text{MakeMeatDish}(n_5) \wedge \\
 &\quad \quad n_5 = n_3 \wedge \text{MakeChickenMarinara}(n_3) \wedge \\
 &\quad \quad \text{step5}(n_3) = n_1 \wedge \text{MakeMarinara}(n_1)))
 \end{aligned}$$

Section 7.2.3 below states the rules for translating an explanation graph into an FOL sentence. The algorithms described in this chapter provide the specialized and *deter-*

ministic proof rules for constructing explanation graphs, given some initial descriptions of event instances.

Each e-graph describes *one* End event. The occurrence of *several* End events is represented by a conjunctively-joined *set* of e-graphs, called an *hypothesis*. To account for all possible ways of *grouping* the observed events, it may be necessary to consider a *disjunctively* joined set of hypotheses. The interpretation of such a set of hypotheses is a sentence true in all minimum covering models of the observations.

7.2. Explanation Graphs

7.2.1. Basic Elements of an E-Graph

An e-graph is made up of various relations over a set of values. A value can be a rigid designator, a fuzzy temporal constraint, or a node.

- Rigid designators are unique names for objects; distinct rigid designators refer to distinct objects. It is a convenient philosophical fiction that proper names like "John Smith" are rigid designators. The rigid designators can be identified with a distinguished subset of FOL constants.

- A time interval can be thought of as a pair of real numbers, representing its beginning and ending instance according to some absolute scale. Yet we can rarely specify a particular time interval in so precise a fashion. Instead, we write expressions which place some relative *constraint* on the interval. Various schemes have been proposed for writing down these constraints, such as Allen's interval operators, which have been used in this thesis up to this point. A special but very useful kind of constraint can be represented by a list of four numbers. Where I is a time interval, and start-min, start-max, end-min, and end-max are numbers on some absolute temporal scale,

$$I \in (\text{start-min}, \text{start-max}, \text{end-min}, \text{end-max})$$

is true when the first instance of I falls between start-min and start-max inclusive, and the final instance of I falls between end-min and end-max inclusion. A large fragment of Allen's interval logic can be translated into an algebra over these "fuzzy" temporal constraints. Appendix D shows how this may be done. The main advantage of this approach is that intervals are all related to a single universal scale, so we do not need to maintain a table relating every interval to every other interval (as in [Allen 1983b]).

Furthermore, disjunctions in the e-graph can lead to disjunctive temporal constraints between *different sets* of intervals, and such disjunctive constraints cannot be handled by a straightforward implementation of Allen's logic.

- A node represents any object which is not given a rigid designator, and in particular, all event instances. Nodes are a distinguished subset of the constants of FOL. Every e-graph includes a function type which maps every node to a unique unary predicate. In particular, every event node is mapped to a member of H_E . An e-graph has exactly *one* node of type End.

7.2.2. Roles of Event Types and of Nodes

The formulation of events described in Chapter 2 treats the roles of an event instance as the unary functions which apply to the instance. Because an event token may be of several types, it is useful to define the subset of roles of an instance which are relevant to each *particular* type of the token. For example, a token :C may be of types Change-Location and Ride-Train. The roles of :C include destination and train-number. While both roles are relevant when one thinks of :C as a Ride-Train, only the former makes sense when one thinks of :C as a Change-Location.

The association of roles with types is an important principle for organizing the construction of an e-graph. Events are abstracted in order to collapse the search space. Some of the roles of the more specialized types of the event are ignored in the process. Unless one "abstracts away" the roles in this manner, it is not possible to have different alternatives for a single abstract event. Consider the example above. The step2 role of N_2 is filled by N_1 . When N_2 is viewed abstractly as N_6 , the step2 role is eliminated. This allows N_6 to also abstract N_3 , which uses N_1 to fill its step5 role. The exact relationship between N_6 and N_1 is left open.

Not all roles of an event are components, of course. Others include the agent of the event, its time, the objects involved, and so on. Any of these roles are also candidates for being abstracted away.

Definitions. The roles of an event type E are the unary functions which appear in the decomposition axiom for any type which abstracts* E, and apply to the variable which appears in the main antecedent of the axiom. The component roles of an event type are the roles of the event type which yield an event token when applied to an event token. The parameters of an event type are all the roles of the event type which are *not* component roles.

In an explanation graph, every node corresponding to an event token has a unique associated type. Therefore we can talk of the pairs of role functions and role values of a node as follows.

Definitions. A roleval of a node N is a pair (f_r, v) such that

1. f_r is a role of $\text{type}(N)$.
2. v is a value, as defined above.
3. $v = f_r(N)$.

The known rolevals of N are simply the rolevals of N for which we can compute the value v . Sometimes we speak of a roleval without reference to a particular node, meaning simply a (f_r, v) pair. The component and parameter rolevals of N are defined in the obvious way.

7.2.3. Definition of an Explanation Graph

An explanation graph $G = (V_r, V_t, V_n, \text{Type}, R, \text{Rolevals}, \text{Alt})$, where:

V_r is the set of rigid designators.

V_t is the set of fuzzy temporal constraints.

V_n is the set of nodes.

Type is a function from nodes to event types.

$$\text{Type}: V_n \Rightarrow H_E$$

R is the set of roles.

Rolevals is the known roleval relation, over node, roles, and values.

$$\text{Rolevals} \subset V_n \times R \times \{V_r \cup V_t \cup V_n\}$$

Alt is the alternatives-for function, from nodes to sets of nodes.

$$\text{Alt}: V_n \Rightarrow 2^{V_n}$$

An interpretation of a node n in G is the following sentence:

$$\begin{aligned}
\text{Interpretation}(n) = & \\
& \{E_i(n) \mid \text{type}(n)=E_i\} \wedge \\
& \wedge \{v=f_r(n) \wedge S \mid (n,f_r,v) \in \text{Rolevals} \wedge v \in V_n \wedge \\
& \quad S = \text{Interpretation}(v)\} \wedge \\
& \wedge \{v=f_r(n) \mid (n,f_r,v) \in \text{Rolevals} \wedge v \in V_r\} \wedge \\
& \wedge \{f_r(n) \in v \mid (n,f_r,v) \in \text{Rolevals} \wedge v \in V_t\} \wedge \\
& \vee \{n=m \wedge S \mid m \in \text{Alt}(n) \wedge S = \text{Interpretation}(m)\}
\end{aligned}$$

The sentence states that n is of the specified type; that it has the specified values as roles; and that it must be equal to one of a set of other nodes. The interpretation of an e-graph is the interpretation of its end node, with nodes replaced by existentially-quantified variables.

$$\begin{aligned}
\text{Interpretation}(G) = & \\
& \exists x_1, \dots, x_k. \text{Interpretation}(h) \theta \\
\text{where} & \\
& h \in V_n \wedge \text{type}(h)=\text{End} \\
& \theta = (n_1/x_1, \dots, n_k/x_k) \\
& \{n_1, \dots, n_k\} = V_n
\end{aligned}$$

It is important to understand that an e-graph, unlike a lexical hierarchy, has existential import. It is used to represent the fact that some particular tokens are of some types, and form components of other tokens. A lexical hierarchy has no existential import. It only asserts that if there are any tokens of some type, then they all must have certain properties.

The treatment of the alternative relation distinguishes e-graphs from previous work that uses "semantic nets" to represent sentences. Systems such as KL-1 [Brachman 85] can only represent conjunctions of atomic assertions. There is no way to assert that a particular entity is *either* of type E_1 *or* of type E_2 . More recent work on semantic nets, such as KRYPTON [Brachman, Fikes, & Levesque 85], has gained the ability to make disjunctive assertions by adding a non-network component which holds sentences containing arbitrary disjunctions and conjunctions. Such systems use a general-purpose theorem prover to make inferences from these assertions. General theorem proving is both theoretically and practically intractable, and little is known about controlling and focusing their power. The special form of an e-graph, however, lets us closely control specific kinds of reasoning with disjunctions.

7.3. Computing the Uses of an Event Type

As we have hinted above, the first stage of the recognition algorithm is to prove that an observed event token is a component of an End event, by considering all the possible *uses* of the observed event. The Uses relation is roughly – but only roughly – the inverse of the direct component relation. Starting from an event of a particular type – say, *GetNewspaper*(C_1) – we consider all the uses of C_1 as a *GetNewspaper* – such as a step of *BecomeInformed*. Then we consider the uses of C_1 for types which abstract and specialize *GetNewspaper*. So if *GetPaper* abstracts *GetNewspaper*, we then consider the uses of C_1 as a *GetPaper*, which *differ* from the uses already considered – such as a step of *BuildFire*. (As will be seen below, we'd actually use different nodes for each abstraction and specialization of C_1 .) Finally, it may be necessary to consider uses of types which specialize the type of the original observation. To allow for the *possibility* that C_1 could be an instance of *GetSleazyTabloid*, we consider the use of the observed event as a step of *EnjoyVicariousTitillation*.

7.3.1. Use Abstraction and Specialization

Just as one can talk of one type abstracting another, one can talk of abstraction of the component relationship between types. In the cooking world, the relationship between *MakeSauce* and *MakePastaDish* abstracts the relationship between *MakeMarinara* and *MakeSpaghettiMarinara*. In order to explain an instance of *MakeMarinara*, one considers *MakeSpaghettiMarinara* and *MakeChickenMarinara*. It is not necessary to consider the path from *MakeMarinara* to *MakeSauce* to *MakePastaDish*, because that way of using the instance abstracts a more specific use of the instance.

A *use* is a triple, (E_c, f_r, E_u) , and stands for the possibility that (some instance) of E_c could fill the f_r role of some instance of E_u . For example, when trying to explain event instance C_1 , this use would mean

$$\exists y . E_u(y) \wedge C_1 = f_r(y) \wedge E_c(C_1)$$

The abstraction relation between uses is defined as follows.

Definition. (E_{ac}, f_r, E_{au}) abstracts (E_c, f_r, E_u) exactly when E_{ac} abstracts E_c , and E_{au} abstracts E_u . When U_1 is either the same as or abstracts U_2 , we say U_1 abstracts* U_2 . Specializes and specializes* are the inverse of abstracts and abstracts* respectively.

7.3.2. The Uses and Direct Component Relations

It is sometimes necessary to consider uses of a type of which that type is *not* a direct component. Consider the following extension to the cooking hierarchy.

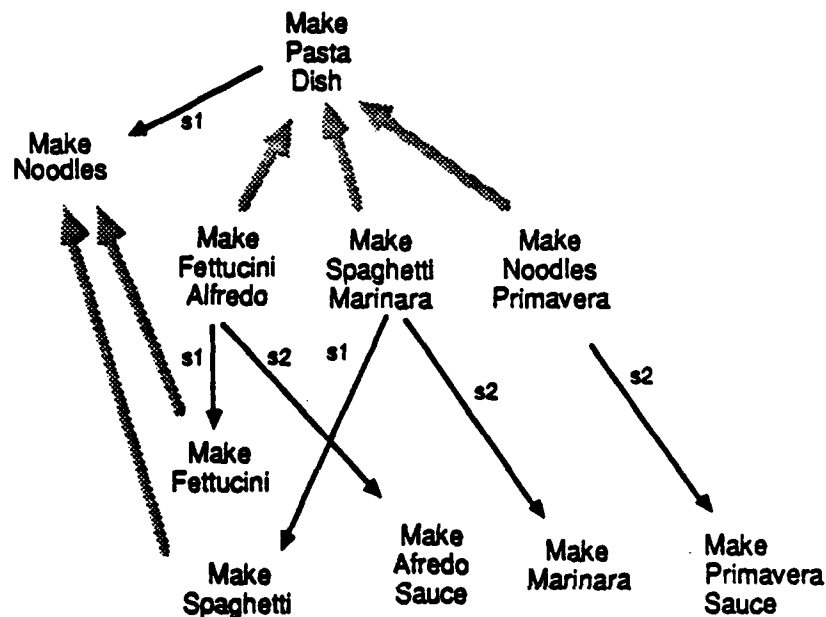


figure 7.2: Extended Cooking Hierarchy Detail

To explain an instance of *MakeSpaghetti*, one considers the use (*MakeSpaghetti*, s_1 , *MakeSpaghettiMarinara*). The use (*MakeNoodles*, s_1 , *MakePastaDish*) is not considered, because it abstracts the former use. But then a possibility is unfairly ignored! An event of type *MakeSpaghetti* could also be a component of an event of type *MakeNoodlesPrimavera*, because that type does not further specify the type of its s_1 role. Therefore the set of Uses to consider should include (*MakeSpaghetti*, s_1 , *MakeNoodlesPrimavera*).

The next section precisely specifies how Uses augments the set of direct components, in order to guarantee that the recognition algorithms are correct. The set Uses needs only be computed once and for all, *before* any observations are input. Therefore we need not be concerned with finding an efficient algorithm to compute Uses; practically any exhaustive, off-line method is sufficient.

7.3.3. Definition of Uses

7.3.4. Example

Following are the uses calculated at each stage for MakeNoodles and MakeSpaghetti, using the extended hierarchy above.

$$U_{\alpha} = \{ \begin{array}{l} (\text{MakeNoodles}, s1, \text{MakePastaDish}) \\ (\text{MakeNoodles}, s1, \text{MakeFettuciniAlfredo}) \\ (\text{MakeNoodles}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeNoodles}, s1, \text{MakeNoodlesPrimavera}) \\ (\text{MakeSpaghetti}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeSpaghetti}, s1, \text{MakePastaDish}) \\ (\text{MakeSpaghetti}, s1, \text{MakeFettuciniAlfredo}) \\ (\text{MakeSpaghetti}, s1, \text{MakeNoodlesPrimavera}) \end{array} \}.$$

Next, eliminate the impossible uses.

$$U_{\beta} = \{ \begin{array}{l} (\text{MakeNoodles}, s1, \text{MakePastaDish}) \\ (\text{MakeNoodles}, s1, \text{MakeFettuciniAlfredo}) \\ (\text{MakeNoodles}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeNoodles}, s1, \text{MakeNoodlesPrimavera}) \\ (\text{MakeSpaghetti}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeSpaghetti}, s1, \text{MakePastaDish}) \\ (\text{MakeSpaghetti}, s1, \text{MakeNoodlesPrimavera}) \end{array} \}$$

Throw out uses where some specializations of the user type are not uses.

$$U_{\chi} = \{ \begin{array}{l} (\text{MakeNoodles}, s1, \text{MakePastaDish}) \\ (\text{MakeNoodles}, s1, \text{MakeFettuciniAlfredo}) \\ (\text{MakeNoodles}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeNoodles}, s1, \text{MakeNoodlesPrimavera}) \\ (\text{MakeSpaghetti}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeSpaghetti}, s1, \text{MakeNoodlesPrimavera}) \end{array} \}$$

Simplify this set by eliminating sets of uses which exhaustively specialize another use.

$$\text{Uses} = \{ \begin{array}{l} (\text{MakeNoodles}, s1, \text{MakePastaDish}) \\ (\text{MakeSpaghetti}, s1, \text{MakeSpaghettiMarinara}) \\ (\text{MakeSpaghetti}, s1, \text{MakeNoodlesPrimavera}) \end{array} \}$$

Thus from MakeNoodles one should consider the use MakePastaDish, while two search paths start from MakeSpaghetti, one through MakeSpaghettiMarinara and the other through MakeNoodlesPrimavera.

7.4. Constraint Checking

Given a component event type, Uses yields a very minimal set of use types which describes those types whose instances could have a component of the given type, but do not *require* the component to be of a *particular* subtype of the given type.

A set of types *describes* another set of types if every member of the latter set *specializes** or *abstracts** a member of the former set. A set of types is *minimal* if no member abstracts another member, and is *very minimal* if it also does not contain a subset $\{E_1, \dots, E_n\}$ which forms an exhaustive set of specializations for some type E_0 . (A very minimal set should contain E_0 instead.) We define Uses in four steps, as follows.

$$U_\alpha = \{ (E_c, f_r, E_u) \mid \exists E_{du}, E_{ac}, E_{au} . \\ E_c \text{ is the } f_r \text{ direct component of } E_{du} \wedge \\ E_{ac} \text{ abstracts* } E_c \wedge \\ E_{ac} \text{ is the } f_r \text{ direct component of } E_{au} \wedge \\ E_{au} \text{ abstracts* } E_u \}$$

$$U_\beta = \{ (E_c, f_r, E_u) \mid (E_c, f_r, E_u) \in U_\alpha \wedge \\ H \cup cl(H) \vdash \forall x . E_u(x) \supset \neg E_c(f_r(x)) \}$$

$$U_\chi = \{ (E_c, f_r, E_u) \mid (E_c, f_r, E_u) \in U_\beta \wedge \\ \forall E_{su} . E_u \text{ abstracts } E_{su} \supset (E_c, f_r, E_{su}) \in U_\beta \}$$

$$\text{Uses} = \{ (E_c, f_r, E_u) \mid (E_c, f_r, E_u) \in U_\chi \wedge \\ \neg \exists E_{au} . E_{au} \text{ abstracts } E_u \wedge (E_c, f_r, E_{au}) \in U_\chi \}$$

This rather complicated construction can be understood as follows. We start by considering E_c which have some explicit direct uses. Then we add uses from E_c to all the types which directly use (in the same role) abstractions of E_c . Next we add uses from E_c to all types which specialize any of the previous uses. (This is to catch uses like (MakeSpaghetti, s_1 , MakeNoodlesPrimavera) above.) This large set is U_α . Next we throw out those uses which are not in fact possible, but arose in the specialization step. This yields U_β . The next two steps redescribe U_β in simpler terms. In U_χ we throw out all the uses where the user type has *some* specializations which *cannot* have components of type E_c . Then in Uses we eliminate a set of uses if they exhaustively specialize a use already under consideration. This corresponds to making uses *very minimal*, as described above.

The algorithms need to check the various constraints which appear in the decomposition axioms for the event types. It is never necessary to prove that a constraint actually holds; rather, one should *fail* to prove that the constraint is false. As a side effect of constraint checking, the values of parameters are propagated to a node from its components. In Section 2.6.3, constraints were categorized as equality constraints, temporal constraints, and preconditions and effects; the latter two jointly called *fact* constraints. Each kind of constraint is handled in a different manner.

7.4.1. Equality Constraints

An equality constraint usually equates a parameter of a node with a parameter of one of its components. An equality constraint fails if the values of the two items are distinct rigid designators. As a side effect, when a value is known for the parameter of the component, but none for the parameter of the node, the value is assigned to the parameter of the node.

7.4.2. Temporal Constraints

Rather than passing around precise values for time parameters, the implementation passes fuzzy temporal constraints. All unknown times are implicitly constrained by

$$(-\infty +\infty \quad -\infty +\infty)$$

A temporal constraint fails if a time parameter is assigned an *empty* fuzzy constraint, such as

$$(5 \ 6 \ 2 \ 3)$$

There can be no time interval which starts between times 5 and 6, and ends between times 2 and 3. (Time doesn't run backwards!) Corresponding to each binary temporal relation, such as Before or During, there an algebraic operation on fuzzy constraints, with the following property:

Where T_1, T_2 are time intervals, and Z_1, Z_2 are fuzzy constraints:

$$T_1 \in Z_1 \wedge T_2 \in Z_2 \wedge T_1 \text{ binary-op } T_2 \supset$$

$$T_1 \in \text{Ftransitive}(Z_1, \text{binary-op}, Z_2)$$

This algebra is described in Appendix D. These rules are used to update the fuzzy constraints assigned to the time parameters of the node under consideration. The appendix

also defines the functions **Fintersection** and **empty**, which are used in the algorithms.

7.4.3. Fact Constraints

Fact constraints are checked only if values are known for all the arguments of the predicates in the constraint. A limited theorem prover attempts to prove the negation of the fact. The constraint fails if the proof succeeds. In order to check a fact of the form

$$\text{Never}(T_1, \text{property})$$

the theorem prover attempts to show that there exists a time T_2 over which the property holds, and T_1 *must* intersect T_2 , in order to satisfy their respective fuzzy temporal constraints.

The implementation assumes that the general domain axioms H_G are such as to guarantee finite failure.

7.5. Overview of the Algorithms

The algorithms are presented in a structured pseudo-code, freely mixing English and programming constructs in order to maximize clarity. All functional parameters are passed by value. There are three basic algorithms: **explain-observation**, **match-graphs**, and **group-observations**. We present an overview of the algorithms, then the pseudo-code, and finally a more detailed description of their operation.

7.5.1. Explain

The function call **explain-observation(etype, edescr)** builds an *e-graph* on the basis of a single observation of an event of *etype*, which satisfies the role/value pairs in *edescr*. Two versions of the subroutine which checks for redundant search paths are presented. The first assumes that the abstraction hierarchy forms a tree. The more general version contains a slightly more complex test to avoid finding redundant paths from a node to End.

Following is an example of the "multi-path problem" with multiple inheritance. Consider the following fragment of an event hierarchy.

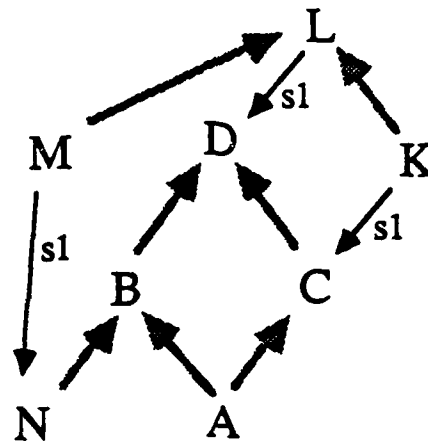


figure 7.3: Multiple Inheritance Hierarchy

Suppose the *primary* event to be explained is of type A. Then one should consider the use (C, s_1 , K) but *not* the more general use (D, s_1 , L). (Why? Because of the exhaustiveness assumption, every L is either an M or a K. M's must have components of type N, and not of type A. Every C is an A. Therefore, from C it is safe to infer K. The weaker conclusion of L is not an *alternative* conclusion.) On the other hand, if the primary node is of type B, then one should consider (D, s_1 , L) but not the more specific (C, s_1 , K). (Why? Because the B could be an N or an A, so the best one can do is conclude L.) The potentially expensive test in the more general version of redundant "looks back" at the primary node to see whether or not a different abstraction path from the primary node could reach a more specific use than the one at the current node.

An important feature of the algorithm is the test involving the consider-spec flag. In order to explain an event of etype, one must consider event types that (1) abstract etype; (2) specialize etype; and (3) abstract types which specialize etype. We must *not* consider types that specialize types that abstract etype – because this includes *all* of H_E .

7.5.2. Match

The function call `match-graphs(graph1, graph2)` builds a new graph which equates the End nodes of graph1 and graph2. The match algorithm assumes that there exists either no or exactly one type which is the "greatest lower bound" of any pair of elements of H_E . Formally:

E_1 and E_2 are compatible $\supset$

$$\begin{aligned} \exists E_3 \in H_E . E_1 \text{ abstracts* } E_3 \wedge E_2 \text{ abstracts* } E_3 \wedge \\ \forall E_4 . E_1 \text{ abstracts* } E_4 \wedge E_2 \text{ abstracts* } E_4 \supset \\ E_3 \text{ abstracts* } E_4 \end{aligned}$$

A technical term for this is that H_E is a lower semi-lattice. The glb of non-compatible types is by definition $\emptyset$.

The key feature of match-graphs is the use of a cache to store matches between nodes. Because e-graphs are digraphs, rather than trees, it is possible for the same pair of nodes to be visited more than once in the matching process. The node created the first time the pair is visited is used as the result of any subsequent match of the pair. Without this feature, match-graphs would multiply the input digraphs into a tree, possibly causing an exponential increase in the size of the result over the size of the inputs.

7.5.3. Group

The function **group-observations** continually inputs observations and groups them into sets to be accounted for by particular e-graphs. The function **minimum-Hypothesis** is called to retrieve a disjunctive set of current hypotheses, each of which is a conjunctive set of e-graphs which accounts for all of the observations, using as few End events as possible.

There are three versions of this algorithm. The Non-Dichronic algorithm returns the same results regardless of the order in which the observations are input, and computes conclusions *mc-entailed* by the inputs. The Incremental Minimization and Sticky versions correspond to the *incremental* theories of recognition discussed in Chapter 5. The Incremental Minimization algorithm only increases the number of different End events under consideration when it has to, and retains all previous conclusions. It implements the *imc-entailment* operator. The Sticky algorithm tries to explain each observation by making it part of the most recently invoked End event.

7.6. Pseudo-Code

7.6.1. Utility Functions

Utility Functions on Nodes of Explanation Graphs

```
function type(node)
    /* most specific type assigned to the node */

function roleval(node, role)
    /* the value of role(node) */

function alternatives-for(node)
    /* a set {n1, n2 ..., nj} such that
       node=n1 ∨ node=n2 ∨ ... ∨ node=nj */

function status(node)
    /* initially True for all nodes, set to False when the alternative con-
       taining the node is ruled out */
```

7.6.2. Explain-Observation

Explain-Observation

```
global egraph /* stores the explanation graph built by explain */

/* explain-observation
   etype : type of observed event
   edescr : role/value pairs of observed event
   returns
       egraph : explanation graph rooted at End-node-of(graph)
   */
function explain-observation(etype, edescr) is
    initialize egraph
    explain(etype, edescr, {etype}, True, etype)
    return egraph
end build-explanation-graph
```

```

/* explain
    etype : type of event to be explained
    edescr : list of (role value) pairs which describe the event
    visited : set of event types considered so far
    consider-spec : if true, then consider specializations of etype
    primary : the type at which search to explain this event
                instance began
returns
    node : represents event token of etype
    pathToEndFound : true if successfully explained event
    allVisited : visited augmented with types considered which
                abstract etype
*/
function explain(etype, edescr, visited, consider-spec, primary) is
    explained := False /* need to find a path to End */
    visited := visited  $\cup$  {etype}
    /* Merge search paths if possible */
    if egraph has a node n of etype which exactly matches edescr then
        return (n, status(n), visited) endif
    /* Can't merge, must consider a new node */
    create a new node n of etype which satisfies edescr in egraph
    if etype=End then return (n, True, visited) endif
    propagate equality, temporal, and fact constraints at n
    if constraints violated then
        status(n) := False
        return (n, False, visited) endif
    /* Consider direct Uses of etype */
    forall (etype, r, utype)  $\in$  Uses
        /* eliminate unnecessary Uses */
        if  $\neg$ redundant(etype, r, utype, visited, primary) then
            ( $\neg$ ,foundpath, $\neg$ ) := explain(utype, {(r,n)},  $\emptyset$ , True, utype)
            /* the r role of the event must be n */
            explained := foundpath or explained
        endif
    endfor
    /* Consider abstractions of etype */
    forall atype  $\in$  direct-abstractions(etype)
        if atype  $\notin$  visited then
            /* eliminate unnecessary abstractions */
            let adescr be the role/value pairs computed for n,
                limited to the non-step roles defined for atype
            (an,foundpath,visited) :=
                explain(atype, adescr, visited, False, primary)
            add n as an alternative for an
            explained := foundpath or explained
        endif
    endfor
endfunction

```

```

        endif
    endif
    /* Consider specializations of etype */
    if consider-spec then
        /* don't abstract up and then specialize back down */
        forall stype ∈ direct-specializations(etype)
            (-,foundpath,-) :=
                explain(stype, edescr, visited, True, primary)
            explained := foundpath or explained
        endfor
    endif
    status(n) := explained
    return (n, explained, visited)
end explain

```

Check for Redundant Paths (called by Explain)

```

/* Tree Version */
function redundant(etype, r, utype, visited, primary) is
    return (etype, r, utype) abstracts or specializes a use
        for some member of visited
end redundant

/* Multiple-Inheritance Version */
function redundant(etype, r, utype, visited, primary) is
    return
        ((etype, r, utype) abstracts a use for a type which
            abstracts* primary) ∨
        ((etype, r, utype) specializes a use for a type which
            specializes* primary) ∨
        (etype does not abstract* or specialize* primary ∧
            (etype, r, utype) specializes a use for a type which
                abstracts primary) ∨
        (etype does not abstract* or specialize* primary ∧
            (etype, r, utype) abstracts a use for a member of
                visited ∧
            (etype, r, utype) does not specialize a use for type
                which abstracts primary)
end redundant

```

7.6.3. Match-Graphs

Match-Graphs

```
global mgraph /* stores the graph created by match */

/* match-graphs
    graph1, graph2 : graphs to be matched
returns
    graph3 : represents equating End nodes of graphs 1 and 2
             and propagating equality
    succeeded : True if match succeeded, false otherwise
*/

function match-graphs(graph1, graph2) is
    initialize mgraph
    (–,succeeded) := match(End-node-of(graph1), End-node-of(graph2))
    return (mgraph, succeeded)
end match-graphs

hash-table previous-match /* cache for results of each call to match */

/* match
    n1 , n2 : values to be matched (either nodes, time intervals, or
             fuzzy temporal constraints)
returns
    n3 : value which represents the result of the match
    succeeded : if true, then n1 and n2 could be equal
*/

function match(n1, n2) is
    if n1 and n2 are rigid designators then
        return (n1, n1=n2)
    elseif n1 and n2 are fuzzy temporal constraints then
        return ( Fintersection(n1,n2), –empty(Fintersection(n1,n2)) )
    else /* n1 and n2 are nodes */
        /* if n1 and n2 have already been matched, reuse that result */
        n3 := previous-match(n1,n2)
        if n3 ≠ ∅ then return (n3, status(n3)) endif
        /* check that types of n1 and n2 are compatible */
        n3type := greatest lower bound of {type(n1), type(n2)}
        if n3type = ∅ then return (–,False) endif
        create new node n3 of n3type in mgraph
        /* add all known role/values of n1 and n2 to n3, matching
           identical roles */
```

```

forall roles r defined for n3type
  v1 := roleval(n1,r)
  v2 := roleval(n2,r)
  If either v1 or v2 is defined then
    If v1 is defined but not v2 then
      (v3, okay) := match(v1,v1)
    elseif v2 is defined but not v1 then
      (v3, okay) := match(v2,v2)
    elseif
      (v3, okay) := match(v1,v2)
    endif
    If ¬okay then return (n3, False) endif
    add v3 as the r role of n3
  endif
endfor
check equality, temporal, and fact constraints on n3
If constraints violated then
  status(n3) := False
  return (n3, False) endif
/* match alternatives for each node */
If either n1 or n2 has an alternative then
  for a1 ∈ alternatives-for(n1)
    (or a1=n1 if no alternatives)
    for a2 ∈ alternatives-for(n2)
      (or a2=n2 if no alternatives)
      (a3, okay) := match(a1, a2)
      If okay then
        add match(a1,a2) as a alternative
        for n3
      endif
    endfor
  endfor
endif
If there were alternatives of n1 or n2, but none matched then
  return (n3,False)
else return (n3,True) endif
endif
end match.

```

7.6.4. Group Observations

7.6.4.1. Non-Dichronic Version

Group-Observations Non-Dichronic Version

global Hypoths

/ A set (disjunction) of hypotheses, each a set (conjunction) of explanation graphs. Each hypothesis corresponds to one way of grouping the observations. Different hypotheses may have different cardinalities. */*

function minimum-Hypothesis **is**

 smallest := min { card(hypo) | hypo ∈ Hypothesis }

return { hypo | hypo ∈ Hypothesis ∧ card(hypo)=smallest }

end minimum-Hypothesis

procedure group-observations **is**

 Hypothesis := {∅}

while more observations

 obtain observation (etype, edescr)

 obs-graph := explain-observation(etype, edescr)

forall hypo ∈ Hypothesis

 remove hypo from Hypothesis

 add hypo ∪ obs-graph to Hypothesis

forall g ∈ hypo

 (new-g, okay) := match-graphs(obs-graph, g)

if okay **then**

 add (hypo - {g}) ∪ new-g to Hypothesis

endif

endfor

endfor

endwhile

end group-observations

7.6.4.2. Incremental Minimization Version

Group-Observations Incremental Minimization Version

```

global Hypoths
    /* A set (disjunction) of hypotheses, each a set (conjunction) of ex-
       planation graphs. Each hypothesis corresponds to one way of
       grouping the observations. All hypotheses have the same cardinali-
       ty, corresponding to the minimum number of End events */

function minimum-Hypothesis is
    return Hypoths
end minimum-Hypothesis

procedure group-observations is
    Hypoths := { $\emptyset$ }
    while more observations
        obtain observation (etype, edescr)
        obs-graph := explain-observation(etype, edescr)
        old-hypothesis := Hypoths
        forall hypo  $\in$  Hypoths
            remove hypo from Hypoths
            forall g  $\in$  hypo
                (new-g, okay) := match-graphs(obs-graph, g)
                if okay then
                    add (hypo - {g})  $\cup$  new-g to Hypoths,
                endif
            endfor
        endfor
        if Hypoths =  $\emptyset$  then
            forall hypo  $\in$  Hypoths
                add obs-graph to hypo
            endfor
        endif
    endwhile
end group-observations

```

7.6.4.3. Sticky Version

Group-Observations Sticky Version

```

global hypo
    /* An ordered list of e-graphs, from most recent to least recent, repre-
       senting a single hypothesis */

```

```

function minimum-Hypothesis is
    return {hypo}
end minimum-Hypothesis

procedure group-observations is
    hypo := NIL
    while more observations
        obtain observation (etype, edescr)
        obs-graph := explain-observation(etype, edescr)
        block update
            foreach g ∈ hypo in order
                (new-g, okay) := match-graphs(obs-graph, g)
                if okay then
                    hypo := cons(new-g, remove(g, hypo))
                    exit block update
                endif
            endfor
            hypo := cons(obs-graph, hypo)
        endblock update
    endwhile
end group-observations

```

7.7. Description of Operation

The following section steps through and explains the operation of the algorithms. The transcripts in Appendix E contain detailed traces of Explain and Match on many of the specific examples discussed in this thesis.

7.7.1. Explain

- Check whether the graph under construction already contains a node of **etype** which exactly matches **edescr**, and contains no additional parameters. If this is the case, then the graph merges at this point. Consider the e-graph at the beginning of this chapter. Suppose the left-hand side of the graph has been constructed (nodes N1, N2, N4, N6, and N7). Search is proceeding along the right-hand part of the graph, through N3. The invocation of **explain** which created **MakeMeatDish(N5)** is considering abstractions of **MakeMeatDish** (see below), and recursively calls **explain** with type **Prepare-Meal**. The specific call would be:

`explain( PrepareMeal, {...}, {MakeMeatDish}, False, MakeMeatDish )`

This description exactly matches previously-created node N6, which is returned. Then N5 is made an alternative for N6. Thus the left path through N2 and N4 merges with the right path through N3 and N5. This kind of merging can prevent combinatorial growth in the size of the graph.

- Create a new node of type *etype*, and link all the role/value pairs in *edescr* to it.
- Check whether the type of the newly-created node is *End*. If so, then no further explanation is possible, and so return.
- Propagate and check constraints. Suppose this is the invocation of *explain* which created *MakeSpaghettiMarinara*(N2). *Edescr* is $\{(\text{step2 } N1)\}$, meaning that component *step2* of N2 is N1. The equality constraints say that the agent of any *MakeSpaghettiMarinara* must equal the agent of its *MakeMarinara* step. (This constraint is inherited from the more general one given for *MakePastaDish*.) If initially (N1 agent Joe) appears in the graph, after this step (N2 agent Joe) also appears.

Fuzzy time constraints are also propagated. N2 is constrained to occur over an interval which contains the time of N1. Suppose the graph initially contains (N1 time (4 5 6 7)). After this step, it also contains (N2 time $(-\infty \ 5 \ 6 \ +\infty)$).

This step can also eliminate nodes, marking them as failed. The agent of every specialization of *MakePastaDish* is constrained to be dexterous. Suppose the general world knowledge base contains the assertion $\neg \text{Dexterous}(\text{Joe})$. Then this step fails for this invocation, and *explain* sets the status of N2 to false and returns.

- Consider Uses of *etype*. The calculation of the set of Uses is described in Section 7.3.3. If *etype* is *MakeMarinara*, then *explain* is recursively invoked with *etypes* of *MakeSpaghettiMarinara* and *MakeChickenMarinara*. The call to *redundant* eliminates uses which have already been considered by alternative paths. Examples follow.
- Explain *etype* by considering its abstractions. The node becomes an *alternative-for* its abstractions. Suppose the current invocation has created *MakePastaDish*(N4). This step calls

`explain(PrepareMeal, {(agent Joe) (time $(-\infty \ 5 \ 6 \ +\infty)$)},
 {MakePastaDish}, False, MakePastaDish)`

which returns N6.

Not all abstractions lead to End; some are pruned, and don't appear in the final graph. Consider the invocation which created MakeMarinara(N1). It calls explain for MakeSauce. The only Use for MakeSauce, however, is (MakeSauce, step2, MakePastaDish); but that use is eliminated by the redundant test. Therefore no node of type MakeSauce appears in the final graph.

- Try to explain etype by considering specializations of etype. This step is not performed if etype was reached by abstracting some other type, rather than as a Use. This always occurs in the example. But suppose explain were initially invoked with etype equal to MakeSauce. Then the specialization MakeMarinara is considered. In the recursive invocation of explain, the Uses of MakeMarinara are examined. The use (MakeMarinara, step2, MakeSpaghettiMarinara) is eliminated by redundant, because it specializes (MakeSauce, step2, MakePastaDish). The use (MakeMarinara, step5, MakeChickenMarinara), however, *does* lead to a path to End.
- If any path to End was found, the status of the node is set to True, and otherwise to False. Finally the name of the node and its status are returned.

7.7.2. Match

The following diagram shows two e-graphs, the first built from an observation of MakeMarinara, and the second from an observation of MakeNoodles. Match is initially invoked on the End nodes of the two graphs:

match(N7, N11)

and returns N12, the End node of the combined graph. The following section steps through the operation of Match.

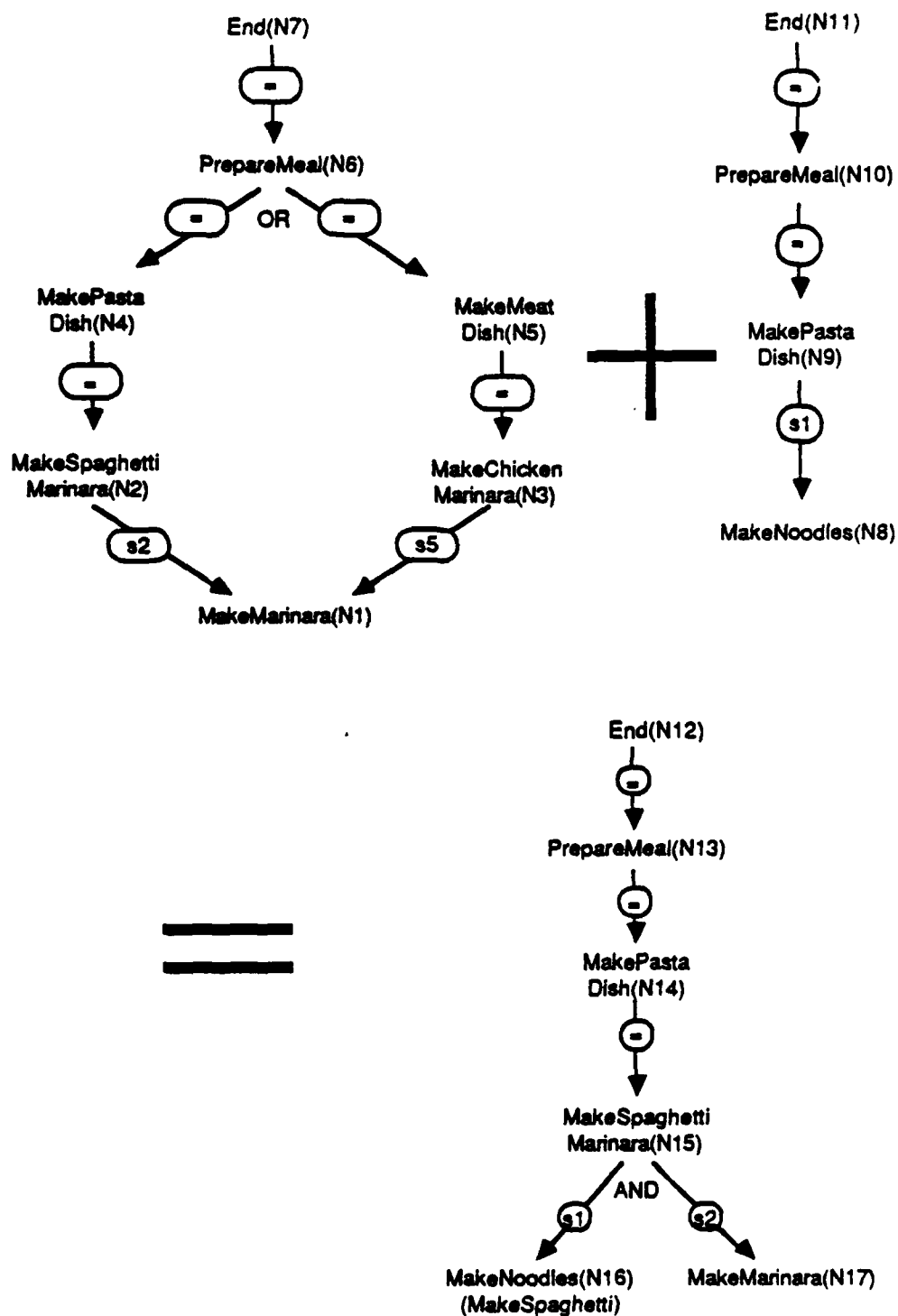


figure 7.4: E-Graphs for *MakeMarinara*, *MakeNoodles*, and their *Match*

- If the objects to be matched are rigid designators, they must be lexically identical.
 - If the objects are fuzzy temporal constraints, then take their intersection. Suppose $n1$ is $(-\infty \ 5 \ 6 \ +\infty)$, and $n2$ is $(-\infty \ 8 \ 7 \ +\infty)$. Then *match* returns $(-\infty \ 5 \ 7 \ +\infty)$.
 - Check whether $n1$ and $n2$ have already been matched, and if so, reuse that value. Suppose that the first e-graph in the diagram above were matched against an e-graph of identical shape; for example, there were two observations of *MakeMarinara*, and they *may* have been identical. During the match down the left hand side of the graphs, through $N4$ and $N2$, *MakeMarinara*($N1$) would match against the *MakeMarinara* node in the second graph (say, $N1'$), resulting in some final node, say $N1''$. Then the right-hand side of the graphs would match, through $N5$ and $N3$. $N1$ would match against $N1'$ a second time, and the value $N1''$ would be used again. This would retain the shape of the digraph, and prevent it from being multiplied out into a tree.
 - Add a new node to the graph, $n3$, to represent the result of the match, which is of the greatest lower bound type of $n1$ and $n2$. Matching *MakeSpaghettiMarinara*($N2$) against *MakePastaDish*($N9$) results in *MakeSpaghettiMarinara*($N15$). The match fails if the types are not compatible. Thus *MakeMeatDish*($N5$) fails to match against *MakePastaDish*($N9$).
 - Match the roles of $n1$ and $n2$. If a role value is defined for one node but not the other, match the role against itself in order to simply copy the structure into the resulting graph. This is what happens when $N2$ matches $N9$, yielding $N15$. The new node gets both the *step2* role value from $N2$ (a copy of $N1$, which is $N17$) and the *step1* role value from $N9$ (a copy of $N8$, which is $N16$). If $N2$ had the role *step1* defined, that value would have had to match against $N8$.
- Suppose both End nodes, $N7$ and $N11$, have the agent roles defined. This step checks that the agents are the same.
- Check and propagate the constraints on $n3$. New constraint violations may be detected at this point because more of the roles of the nodes are filled in. Violations of temporal orderings between steps are detected at this point. The transcripts contain an example where an e-graph based on *Boil* fails to match against one based on *MakeNoodles*, because the *Boil* occurred *before* the *MakeNoodles*.
 - Try matching every alternative for $n1$ against every alternative for $n2$. The successful matches are alternatives for $n3$. If one of the nodes has some alternatives, but the other does not, then match the alternatives for the former against the latter directly.

This occurs in the example above. MakePastaDish(N4) matches against MakePastaDish(N9). N4 has the alternative N2, but N9 has none. Therefore MakeSpaghettiMarinara(N2) matches against MakePastaDish(N9) as well.

If there were some alternatives but all matches failed, then n3 fails as well.

- Return n3 and whether or not the match succeeded.

7.7.3. Group

The various versions of group-observations are considerably simpler than the previous algorithms.

The Non-Dichronic Version maintains a set of all possible groupings of the observations. It functions as follows:

- Observe an event of etype satisfying edescr.
- Generate obs-graph by explain.
- *Conjoin obs-graph* with each hypothesis consisting of a set of e-graphs. Thus obs-graph is considered to be *unrelated* to the previous observations.
- Try to match obs-graph with each e-graph in each hypothesis. Thus obs-graph is considered to be *related* to a previous observation.
- The current (mc-entailed) conclusion corresponds to the *disjunction* of all hypotheses of minimum size.

The Incremental Minimization Version works as follows:

- Observe an event of etype satisfying edescr.
- Generate obs-graph by explain.
- Try to match obs-graph with each e-graph in each hypothesis. Thus obs-graph is considered to be *related* to a previous observation. Each successful match leads to a new hypothesis.

- If **obs-graph** *cannot* be matched with previous e-graphs, then *conjoin* **obs-graph** with each hypothesis.
- The current (imc-entailed) conclusion corresponds to the *disjunction* of all hypotheses, all of which are of same size.

Finally, the Sticky Version works as follows:

- Observe an event of etype satisfying edescri.
- Generate **obs-graph** by explain.
- Try to match **obs-graph** with each e-graph in the (single) hypothesis, in chronological order, from most to least recent. As soon as a match succeeds, update the hypothesis, removing the old matching e-graph, and adding the result of the match as the most recent e-graph.
- If **obs-graph** *cannot* be matched with any previous e-graphs, then *conjoin* **obs-graph** with the hypothesis, as the most recent e-graph.
- The current conclusion corresponds to the single hypotheses, a conjunction of e-graphs.

7.8. Completeness & Correctness

The necessarily limited nature of the constraint checking performed by the algorithms, together with the open-ended nature of the plan recognition problem, makes it impossible to guarantee that the algorithms will arrive at the *strongest* conclusions justified by the theory of mc-entailment. The algorithms are, however, *correct*: the interpretation of their outputs holds in all minimum-covering models. This most easily seen by noting that the algorithms implement the proof rules discussed in Chapters 3 and 4.

The key to the correctness of **explain** is the fact that the algorithm searches from the input event type to every possible use of events of that type, as specified by Theorem 3.11. (A precise statement and proof of this fact, which demonstrates that the set *Uses* and the *redundant* check are properly formulated, appears in Appendix F.) Each path leads to a different disjunctive conclusion, which may be collapsed with others via the *alternative-for* arcs.

The match algorithm simply propagates an equality assertion between terms. If two terms $n1$ and $n2$ are equal, then the value of a function applied to the first is the same as the value of that function applied to the second. If $n1$ is equal to one of a set of alternatives, and $n2$ is equal to one of another set of alternatives, then certainly at least one alternative from the first set is equal to some alternative from the second. The Disjointedness Assumptions justify concluding that $n1$ and $n2$ are not equal if they are not of compatible types.

7.9. Complexity

One straightforward measure of the complexity of the algorithms is the size of largest data structure possibly produced. A more detailed analysis would measure time complexity as well; but since the algorithms visit each node a fixed number of times, we can take space complexity as a rough measure of time complexity as well. (A possible exception is the test to handle the multi-path problem in the multiple-inheritance version of explain-observation. The cost of the best algorithm to perform this test is not obvious.)

The size of an e-graph, $|G|$, is the number of nodes in G . The expression "Algorithm X is of $O(f(n))$ " in the following sections means that the largest data structure constructed by X on input n is no larger than $c*f(n)$, for some fixed constant c . The actual worst-case complexity of X may, of course, be smaller than $f(n)$. The expression "Algorithm X is no better than $O(f(n))$ " means that there is no g such that X is of $O(g(n))$ and g is better than f by more than a constant factor; that is, it is not the case that

$$\forall k > 0 . \exists m . \forall n . n > m \supset g(n) < k*f(n)$$

7.9.1. Explain

Suppose that the events described by H contain no parameters. Then explain-observation is of $O(|H_E|)$, since the worst case would lead to an exhaustive search of all of H . But suppose explain-observation did not merge search paths at abstraction points. Then the algorithm would be no better than $O(2^{|H_E|})$, because an event can have several different components of the same type. The figure below is a case where such combinatorial blowup could occur.

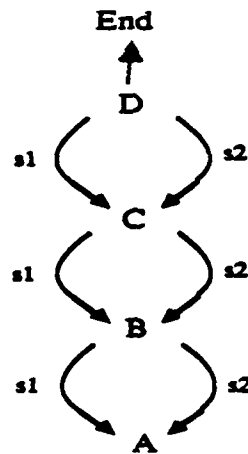


figure 7.5: Combinatorially Explosive Hierarchy

This example shows the importance of maintaining disjunctions within each e-graph, using the alternative-for relation, rather than creating separate databases for each alternative.

If parameters (such as agent, time, etc.) do appear, the search space will be larger. During the construction of the e-graph, however, very few of these roles will have known values. Thus the first step in explain will very frequently find a similar node and cut off search.

7.9.2. Match

Calling $\text{match-graphs}(G_1, G_2)$ frequently returns a graph which is smaller than either G_1 or G_2 . Unfortunately, sometimes the resulting graph can be larger. This occurs when two nodes are matched which have several alternatives, all of which are mutually compatible. Therefore the worst case complexity of $\text{match-graphs}(G_1, G_2)$ is $O(|G_1| * |G_2|)$.

There cannot be more than $|G_1| * |G_2| + |G_1| + |G_2|$ nodes in the result, because every pair of nodes always matches to the same node. The last two addends enter in because a node may also match against itself. Without the use of the cache of previous results, performance could be much worse. The resulting graph would always be a tree, and the algorithm would be no better than $O(2^{|G_1| * |G_2|})$.

7.9.3. Group

Some of the most intimidating complexity results arise from the group-observations algorithm. The non-dichronic algorithm finds all consistent *partitions* of the set of observations. In the worst case, all partitions are possible – any subset of the observed events could be part of the same End event. Then Hypoths will contain $B(n)$ hypotheses, where n is the number of observations. There is no simple closed form for $B(n)$, but it is easy to show that

$$B(n) < \frac{n^n}{n!} = \frac{n^n}{\left(\frac{n}{e}\right)^n \sqrt{2\pi n}} = \frac{e^n}{\sqrt{2\pi}}$$

so that group-observations could contain at least 2^n hypotheses.

Together with the worst-case estimates for explain and match, Hypoths could total more than $O(2^n |H_E|^n)$ nodes. Of course, if all the observations could be part of the same End event, then minimum-hypos will return a single hypothesis consisting of a single e-graph which incorporates all the observations.

The basic dichronic algorithm does much better in this case. If all the observed events could be part of the same hypothesis, then Hypoths will be of size 1. This worst-case size of this hypothesis is $O(|H_E|^n)$. This may not seem like much of an improvement, but remember that the $|H_E|^n$ is a very unlikely upper bound – it would only hold if all the observations were not telling us *anything* definite! – while the 2^n factor in the non-dichronic algorithm *would be accurate* in the common case of all the observations being possibly related.

Once several End events are required to account for all the observations, however, the dichronic algorithm could fall prey to the same sort of combinatorial explosion. Thus a truly practical system may need to resort to the sticky dichronic algorithm. Because it develops a single hypothesis, the sticky algorithm has an absolute upper bound of $O(|H_E|^n)$ (and will usually do much better). Sometimes the sticky algorithm will fail to find the correct – or any – interpretation of the data. An endless number of more or less general and more or less efficient algorithms lie between the non-dichronic and sticky versions.

7.10. Transcripts

The explain-observation and match-graphs algorithms have been implemented in Common Lisp, and appear practical in small domains. The recognition problems discussed in this thesis all run in a few seconds. Appendix E contains transcripts of the programs in operation.

Chapter 8

Conclusions

8.1 The Three Levels

[Marr 82] argues that research in A.I. should analyze each problem at three distinct levels. The *computational theory* states the goal of the computation, why it is appropriate, and provides an abstract mapping from the input information to the output. The level of *representation and algorithms* provides a particular representation for the input and output, and an algorithm for the transformation. The *hardware implementation* shows how the algorithm can be realized physically. Quite often the computational level is overlooked; the result, as Marr demonstrates with many examples from the study of vision, are algorithms which do not actually solve the intended problem, and are based on incorrect assumptions. Neglecting the algorithmic theory is dangerous in another way: one can develop elegant formal theories which would require infinite time and resources to implement.

Our framework includes model, proof, and algorithmic theories. How do these relate to the three levels? The model theory provides a mapping from models of the input to models of the conclusions; thus it is a pure information to information mapping, a computational level theory of the most abstract kind. The proof theory is a computational theory of a more concrete kind. It provides a non-deterministic procedure (namely, the generation of the non-monotonic assumptions, followed by the application of deductive rules) that transforms a first-order representation of the input to that of the output. Thus the proof theory stands between Marr's computational and algorithmic levels. Finally our algorithmic theory fills the algorithmic level, by providing a deterministic procedure which may be directly programmed on a computer. We have nothing to say about the hardware level, how anything like our theory could be realized in a brain. Practically all work that suggests how high-level inference could be performed by neurons is fueled mostly by (often ingenious) speculation; but see [Feldman 81] for some proposals in this direction.

Having completed this thesis, there is no doubt in my mind that the most difficult task was to define the computational theory at the most abstract level. Once the rather amorphous problem of "plan recognition" was tied down, work could proceed in earnest. As the literature review in Chapter 1 revealed, similar proposals and heuristics for plan recognition problems have been hashed over for years, since the early work of

Schmidt and Genesereth. The minimal model construct we've developed suggests what some of these heuristics are heuristics *for*. The lack of a clear problem statement is no doubt one of the reasons for the stagnation in work in story understanding in the past decade.

8.2 Applicability

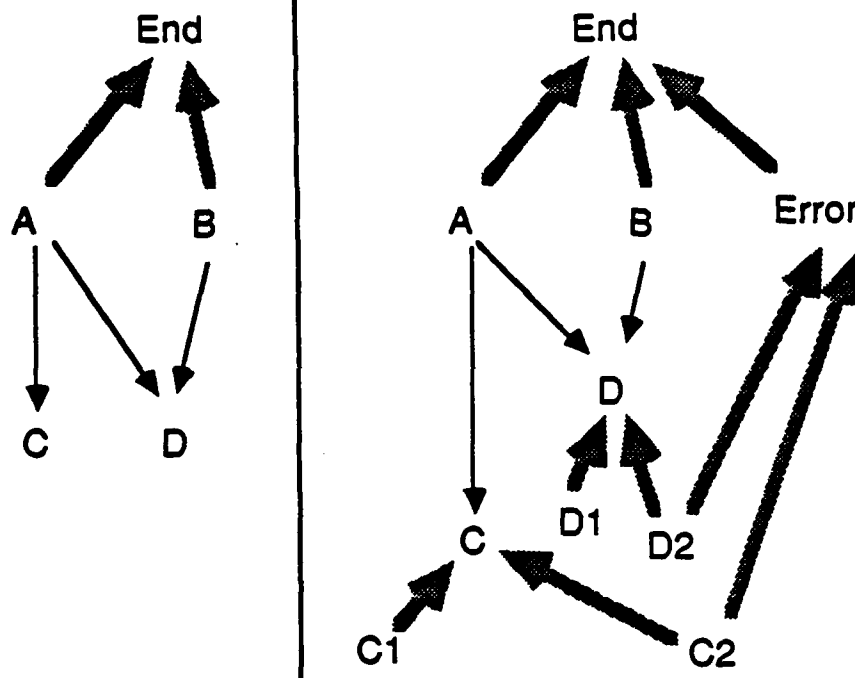
Our framework makes explicit the assumptions of Exhaustiveness, Disjointedness, and Component/Use Completeness which underlie our theory of plan recognition, as well as most previous work. These assumptions simplify the problem, and restrict its applicability. Do these assumptions limit our work to toy micro-worlds?¹

Every day new kinds of events occur, and yet they do not baffle us. An intelligent agent cannot rely only on a recognition system; it must contain a learning component as well. Earlier we suggested that it might be appropriate to invoke the learning module when recognition failed. The ability of our framework to handle levels of abstraction provides a crucial feature needed for learning. For example, we might be able to conclude that the agent is cooking some kind of pasta dish, even if we can rule out any of the *particular* pasta dishes in our library. It may be possible to modify the minimal model construction so that the Exhaustiveness assumption is weakened, and thus allow the recognition and learning modules to be integrated.

None the less, much of what a plan recognition system must handle is routine. A more serious problem is that of recognizing *erroneous* plans. Our framework assumes all plans are internally consistent, and that all acts are purposeful. Yet real people make frequently make planning errors and change their minds in midcourse.

One class of errors can be handled without significant change to our framework. Suppose *every* basic event type is given two new specializations, one of which also specializes a new type called Error. The diagram below illustrates how a hierarchy would be transformed.

¹[Dreyfus 85] argues that this criticism applies to work in A.I. in general.



Then each observed event could either be an End event on its own by being an Error, or it could be part of an End event in a correct plan from the original hierarchy. Minimizing the number of End events prefers interpretations in which Errors do not occur; but observations sequences which *must* be part of erroneous plans can still be recognized.

8.3 Generality & Extensibility

Despite the limitations outlined above, the generality of our framework must be stressed. We do *not* assume that there is a single plan underway, which can be uniquely identified from the first input. We do *not* assume that the sequence of observations is complete.² Finally, we do *not* assume that all the steps in a plan are linearly ordered. Indeed, we know of no other implemented plan recognition system which handles arbitrary temporal relations between steps.

The logical basis for the system makes it clear what conclusions should be reached on the basis of quantified and disjunctive information. Section 2.7 showed

²Completeness of the observations is a nice property to exploit when it is available. [Segre 86] has built a plan recognition system for use in programming a robot arm. The operator guides the arm manually, and the system recognizes what plans are being performed. No kind of non-deductive inference is needed: because every step of the plan is input, the fact that the plan occurs logically follows.

how plans involving conditional actions could be represented by event hierarchies. Previous plan recognition systems did not handle any of these situations.

The generality of the system suggests that it may be extended in various ways in a straightforward manner. An important one for plan recognition is the ability to deal with plans containing iteration and recursion. For example, a plan to pick up all the blue blocks might be encoded as follows:

$$\forall x . \text{pickup-blues}(x) \supset \forall y . \text{block}(y) \wedge \text{blue}(y) \supset \\ \text{pickup}(\text{s}(y,x)) \wedge \text{object}(\text{s}(y,x))=y$$

That is, if an instance of picking up blue blocks occurs, then for every block that is blue, an instance of pickup occurs with that block as its object. We have not yet investigated whether this extension to the form of the event hierarchy requires modification in the model theory. Another area for future work is to extend our notion of an event hierarchy to include hierarchies with exceptions [Touretzky 84].

Finally we need to consider principles other than Occam's Razor for determining the class of preferred interpretations. Rather than ordering models during the minimization process by the number of End events, one may wish to include other factors, such as qualitative likelihoods of various event types. It remains to be seen how far the general framework can be pushed, or if other, more quantitative measures belong in a separate theory, which applies to the *conclusions* of our system.

8.4 Two Unresolved Issues

After reviewing the strong and weak points of our work, a couple of unresolved issues remain. The first is the question of scale-up. As discussed in Chapter 7, the worst-case behavior of our recognition algorithms can be bad, although proper structuring of the hierarchy greatly reduces search. How much of a problem will this be in more realistically-sized knowledge bases? Experience in creating and examining the structure of larger domains is necessary.

A more philosophical problem is the whole issue of what serves as primitive input to the recognition system. Throughout this thesis we've assumed that arbitrary high-level descriptions of events are simply presented to the recognizer. This assumption is reasonable in many domains, such as understanding written stories, or observing the words typed by a computer operator at a terminal. But a real plan recognizer – a person – doesn't always get his or her input in this way. How are visual impressions

of simple bodily motions – John is moving his hands in such and such a manner – translate into the impression that John is rolling out dough to make pasta? There is a great deal of work in low-level perception, and a great deal in high level recognition – including this thesis. The semantic gap between the output of the low-level processes and the high-level inference engines remains wide, and few have ventured to cross it.

References

- Allen, James (1983) Recognizing Intentions From Natural Language Utterances, in *Computational Models of Discourse*, eds. Michael Brady & Robert Berwick, The MIT Press, Cambridge.
- Allen, James (1983b) Maintaining Knowledge About Temporal Intervals, *Communications of the ACM*, no. 26, pp. 832-843.
- Allen, James (1984) Towards a General Theory of Action and Time, *Artificial Intelligence*, vol. 23, no. 2, pp. 832-843.
- Allen, James & Alan Frisch (1982) What's In a Semantic Network?, *Proceedings of the ACL-82*, Toronto.
- Allen, James & Johannes Koomen (1983) Planning Using a Temporal World Model, *Proceedings of IJCAI-83*, Karlsruhe, West Germany.
- Allen, J.F. & C.R. Perrault (1980) Analyzing Intention in Dialogues, *Artificial Intelligence*, vol. 15, no. 3, pp. 143-178.
- Barwise, Jon (1977) editor, *The Handbook of Mathematical Logic*.
- Blenko, Tom (1985) Suggestions and Offers, in *Report on Commonsense Summer*, ed. Jerry Hobbs, Center for the Study of Language and Information, Stanford University.
- Bossu, G. & P. Siegel (1985) Saturation, Non-Monotonic Reasoning, and the Closed-World Assumption, *Artificial Intelligence*, vol. 25, no. 1, pp. 13-63.
- Brachman, Ronald J. (1985) On the Epistemological Status of Semantic Nets, in *Readings in Knowledge Representation*, eds. R.J. Brachman & H.J. Levesque, Morgan Kaufman, Los Altos, CA.
- Brachman, Ronald J., Richard E. Fikes & Hector J. Levesque (1985) KRYPTON, A Functional Approach to Knowledge Representation, in *Readings in Knowledge Representation*, eds. R.J. Brachman & H.J. Levesque, Morgan Kaufman, Los Altos, CA.
- Bruce, B.C. (1981) Plans and Social Action, in *Theoretical Issues in Reading Comprehension*, eds. R. Spiro, B. Bruce, & W. Brewer, Lawrence Erlbaum, Hillsdale, New Jersey.
- Carberry, Sandra (1983) Tracking Goals in an Information Seeking Environment, *Proceedings of AAAI-83*, Washington, D.C.
- Carbonell, Jaime (1979) The Counterplanning Process: A Model of Decision-Making in Adverse Situations, Technical Report (unnumbered), Computer Science Department, Carnegie-Mellon University.
- Charniak, Eugene & Drew McDermott (1985) *Introduction to Artificial Intelligence*, Addison Wesley, Reading, MA.

- Clark, K.L. (1978) Negation as Failure, in *Logic and Databases*, ed. J. Minker, Plenum Press, New York.
- Cohen, P.R. (1978) On Knowing What to Say: Planning Speech Acts, TR 118, Department of Computer Science, University of Toronto, Ontario.
- Cohen, Phillip (1984) Referring as Requesting, *Proceedings of COLING-84*, pp. 207, Stanford University.
- Cohen, P. & Levesque, H. (1980) Speech Acts and the Recognition of Shared Plans, *Proceedings of the CSCSI-80*, Victoria, British Columbia, pp. 263-271.
- Cohen, P. & Levesque, H. (1987) Rational Interaction as the Basis for Communication, *Proceedings of the Symposium on Intentions and Plans in Communication and Discourse*, Center for the Study of Language & Information, Stanford, CA.
- Cohen, P., R. Perrault, & J. Allen (1981) Beyond Question-Answering, Report No. 4644, BBN Inc., Cambridge, MA.
- Davis, Martin (1980) The Mathematics of Non-Monotonic Reasoning, *Artificial Intelligence*, vol. 13, pp. 73-80.
- Dreyfus, Hubert L. (1985) From Micro-Worlds to Knowledge Representation: A.I. at an Impasse, in *Readings in Knowledge Representation*, eds. R.J. Brachman & H.J. Levesque, Morgan Kaufman, Los Altos, CA.
- Dwyer, M.G. (1982) In Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension, TR 219, Department of Computer Science, Yale University.
- Etherington, David (1986) Reasoning with Incomplete Information: Investigations of Non-Monotonic Reasoning, Technical Report 86-14, Department of Computer Science, University of British Columbia, Vancouver, Canada.
- Etherington, David (1987) personal communication.
- Feldman, Jerome A. (1981) Memory and Change in Connection Networks, TR 96, Department of Computer Science, University of Rochester, Rochester, NY.
- Fillmore, Charles (1968) The Case for Case, in *Universals in Linguistic Theory*, Holt, New York.
- Genesereth, Michael (1979) The Role of Plans in Automated Consulting, *Proceedings of IJCAI-79*, pp. 119.
- Genesereth, Michael (1983) An Overview of Meta-Level Architecture, *Proceedings of AAAI-83*, Washington, D.C.
- Goldman, Alvin (1970) *A Theory of Human Action*, Princeton University Press, Princeton.

- Green, C. (1969) An Application of Theorem Proving to Problem Solving, *Proceedings of IJCAI-69*, pp. 219-239, Washington, D.C.
- Grosz, B.J. (1977) The Representation and Use of Focus in Dialogue Understanding, Technical Note 151, SRI International, Palo Alto.
- Harel, David (1979) *First-Order Dynamic Logic*, Springer-Verlag.
- Hacking, Ian (1983) *Representing and Intervening: Introductory Topics in the Philosophy of Natural Science*, Cambridge University Press.
- Hayes, P.J. (1985) The Logic of Frames, in *Readings in Knowledge Representation*, eds. R.J. Brachman & H.J. Levesque, Morgan Kaufman, Los Altos, CA.
- Hintikka, Jaakko (1962) *Knowledge and Belief*, Cornell University Press, Ithaca, NY.
- Huff, Karen & Victor Lesser (1982) KNOWLEDGE-BASED COMMAND UNDERSTANDING: An Example for the Software Development Environment, TR 82-6, Computer and Information Sciences, University of Massachusetts at Amherst.
- Kunz, John C. (1983) Analysis of Physiological Behavior Using a Causal Model Based on First Principles, *Proceedings of AAAI-83*, Washington, D.C., pp. 225-228.
- Kautz, Henry (1985) Towards a Theory of Plan Recognition, TR 162, Department of Computer Science, University of Rochester.
- Kautz, Henry & James Allen (1986) Generalized Plan Recognition, *Proceedings of AAAI-86*, Philadelphia.
- Kautz, Henry (1986b) The Logic of Persistence, *Proceedings of AAAI-86*, Philadelphia.
- Kautz, Henry (1986c) Poèmes humoristiques sur l'AI, *Canadian Artificial Intelligence*, no. 9, Sept.
- Kautz, Henry (1987) *A Formal Theory of Plan Recognition*, PhD Thesis, Department of Computer Science, University of Rochester.
- Kyburg, Henry (1974) *The Logical Foundations of Statistical Inference*, D. Reidel, Dordrecht-Holand.
- Lehnert, Wendy (1980) Affect Units and Narrative Summarization, TR 179, Department of Computer Science, Yale University.
- Lewis, David (1969) *Convention*, Harvard University Press, Cambridge.
- Lifschitz, Vladimir (1984) Some Results on Circumscription, *Proceedings of the AAAI Workshop on Non-Monotonic Reasoning*.

- Litman, Diane & James Allen (1984) A Plan Recognition Model for Clarification Sub-dialogues, TR 141, Department of Computer Science, University of Rochester, NY.
- Marr, David (1982) *Vision*, W.H. Freeman, New York.
- McAllester, David (1980) An Outlook on Truth Maintenance, AI Memo 551, M.I.T..
- McCarthy, John (1980) Circumscription -- A Form of Non-Monotonic Reasoning, *Artificial Intelligence*, vol. 13, pp. 27-39.
- McCarthy, John (1984) Applications of Circumscription to Formalizing Common Sense Knowledge, *Proceedings of the AAAI Workshop on Non-Monotonic Reasoning*.
- McCarthy, J. & P. Hayes (1969) Some Philosophical Questions from the Standpoint of Artificial Intelligence, *Machine Intelligence 4*, Edinburg University Press, Edinburg, UK.
- McDermott, Drew (1981) A Temporal Logic for Reasoning About Processes and Plans, Research Report #196, Department of Computer Science, Yale University.
- Minker, J. & Perlis, D. (1985) Circumscription: Finitary Completeness Results, Technical Report, Computer Science Department, University of Maryland.
- Minsky, Marvin (1975) A Framework for Representing Knowledge, in *The Psychology of Computer Vision*, McGraw-Hill, New York.
- Nilsson, N.J. (1980) *Principles of Artificial Intelligence*, Tioga Press, Palo Alto.
- Patil, Ramesh, Peter Szolovits, & William Schwartz (1982) Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders, in *Artificial Intelligence in Medicine*, ed. Peter Szolovits, AAAS Select Symposium 51, Westview Press, Boulder, Colorado.
- Peirce, Charles S. (1958) *Collected Papers of Charles Sanders Peirce*, Cambridge, Mass.
- Pereira, F. & D. Warren (1982) Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Transition Networks, *Artificial Intelligence*, vol 13, pp 231-278.
- Perlis, D. (1980) *Truth, Syntax, and Reason*, PhD Thesis, Computer Science Department, University of Rochester.
- Pierrehumber, Janet & Julia Hirschberg (1987) The Meaning of Intonational Contours in the Interpretation of Discourse, in *Readings in Knowledge Representation*, eds. R.J. Brachman & H.J. Levesque, Morgan Kaufman, Los Altos, CA.
- Pollack, Martha (1986) A Model of Plan Inference that Distinguishes Between the Beliefs of Actors and Observers, *Proceedings of the ACL-86*, New York.

- Pople, Harry (1973) On the Mechanization of Abductive Logic, *Proceedings of IJCAI-73*, Stanford University, CA.
- Pople, Harry (1977) The Formation of Compound Hypotheses in Diagnostic Problem Solving: an Exercise in Synthetic Reasoning, *Proceedings of IJCAI-77*, Cambridge, MA.
- Pople, Harry (1982) Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics, in *Artificial Intelligence in Medicine*, ed. Peter Szolovits, AAAS Select Symposium 51, Westview Press, Boulder, Colorado.
- Reggia, J., D.S. Nau, & P.Y. Wang (1983) Diagnostic Expert Systems Based on a Set Covering Model, *International Journal of Man-Machine Studies*, vol. 19, pp. 437-460.
- Reichman, R. (1981) Plan Speaking: A Theory and Grammar of Spontaneous Discourse, Report 4681, BBN Incorporated, Cambridge.
- Reiter, R. (1980) A Logic for Default Reasoning, *Artificial Intelligence*, vol. 13, no. 2.
- Reiter, Ray (1982) Circumscription Implies Predicate Completion (Sometimes), *Proceedings of AAAI-82*, William Kaufman, Inc.
- Rich, Charles (1981) Inspection Methods in Programming, AI-TR-604, Laboratory for Artificial Intelligence, M.I.T..
- Robinson, G. & L. Wos (1969) Paramodulation and Theorem-Proving in First-Order Theories with Equality, *Machine Intelligence 4*, pp. 135-150, Edinburg University Press, Edinburg, UK.
- Sacerdoti, Earl (1977) *A Structure for Plans and Behavior*, Elsevier, New York.
- Schafer, Glenn (1976) *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ.
- Schank, R. (1975) *Conceptual Information Processing*, American Elsevier, New York.
- Schank, Roger (1983) The Current State of A.I.: One Man's Opinion, *The A.I. Magazine*, vol. 4, no. 1.
- Schmidt, C.F., N.S. Sridharan, & J.L. Goodson (1978) The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence, *Artificial Intelligence*, vol. 11, pp. 45-83.
- Schmolze, James G. (1986) Physics for Robots, *Proceedings of AAAI-86*, Philadelphia.
- Segre, Alberto (1976) Talk given at University of Rochester.

- Sidner, Candace (1983) Focusing in the Comprehension of Definite Anaphora, in *Computational Models of Discourse*, eds. M. Brady & R. Berwick, M.I.T. Press, Cambridge.
- Sidner, Candace (1985) Plan Parsing for Intended Response Recognition in Discourse, *Computational Intelligence*, vol.1, no. 1, pp. 1.
- Sidner, Candace & David Israel (1981) Recognizing Intended Meaning and Speakers' Plans, *Proceedings of IJCAI-81*, University of British Columbia, Vancouver, BC, pp. 203-298.
- Smith, Reid G. & Randall Davis (1981) A Framework for Cooperation in Distributed Problem Solving, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11, no. 1.
- Stefik, Mark (1980) Planning with Constraints, Report STAN-CS-80-784, Department of Computer Science, Stanford University.
- Tenenburg, Josh (1986) Reasoning Using Exclusion: An Extension of Clausal Form, TR 147, Department of Computer Science, University of Rochester.
- Vilain, Marc & Henry Kautz (1986) Constraint Propagation Algorithms for Temporal Reasoning, *Proceedings of AAAI-86*, Philadelphia.
- Wilensky, Robert (1982) Talking to UNIX in English: An Overview of U.C., *Proceedings of AAAI-82*, Carnegie-Mellon University, Pittsburgh.
- Wilensky, Robert (1983) *Planning and Understanding*, Addison-Wesley, Reading, MA.

Appendix A

Chapter 3 Proofs

Theorem 3.1 (Exhaustiveness)

Suppose $\{E_1, E_2, \dots, E_n\}$ are all the predicates directly abstracted by E_0 in H_A . Then the statement:

$$\forall x . E_0(x) \supset (E_1(x) \vee E_2(x) \vee \dots \vee E_n(x))$$

is true in all models of H_A which are closed under specialization. The statement is also true in all A-closed models of H.

Proof

Let M_1 be a model of H_A in which the statement does not hold. We prove that M_1 cannot be closed under specialization.

For the statement to be false in M_1 , there must be some $:C$ such that

$$E_0(x) \wedge \neg E_1(x) \wedge \dots \wedge \neg E_n(x)$$

is true in $M_1\{x/:C\}$. Define M_2 by

$$\text{Domain}(M_2) = \text{Domain}(M_1)$$

$$M_2[Z] = M_1[Z] \text{ for } Z \neq E_0$$

$$M_2[E_0] = M_1[E_0] - \{ :C \}$$

That is, M_2 is the same as M_1 , except that $:C \notin M_2[E_0]$.

We claim that M_2 is a model of H_A . Every axiom that does not contain E_0 is true in M_1 , and therefore receives the same valuation in M_2 . So consider axioms that contain E_0 . Axioms of the form:

$$\forall x . E_0(x) \supset E_i(x) \quad i \neq 0$$

are false only if there is a $:D$ such that

$$:D \in M_2[E_0] \wedge :D \notin M_2[E_i]$$

But we know that

$$\begin{aligned}
& :D \in M_2[E_0] \Rightarrow \\
& :D \in M_1[E_0] \Rightarrow \\
& :D \in M_1[E_i] \Rightarrow \\
& :D \in M_2[E_i]
\end{aligned}$$

which is a contradiction. Therefore there can be no such $:D$, and the axiom holds in M_2 .

The only other axioms containing E_0 are of the form

$$\forall x . E_i(x) \supset E_0(x) \quad 1 \leq i \leq n$$

Again, this fails in M_2 if there is a $:D$ such that

$$:D \in M_2[E_i] \wedge :D \notin M_2[E_0]$$

If $:D \neq :C$, then

$$\begin{aligned}
& :D \in M_2[E_i] \Rightarrow \\
& :D \in M_1[E_i] \Rightarrow \\
& :D \in M_1[E_0] \Rightarrow \\
& :D \in M_2[E_0]
\end{aligned}$$

which is a contradiction. Finally, if $:D = :C$, then, since $:C \notin M_1[E_i]$,

$$\begin{aligned}
& :D \notin M_1[E_i] \Rightarrow \\
& :D \notin M_2[E_i]
\end{aligned}$$

which also is a contradiction. Therefore there can be no such $:D$. So M_2 is a model of H_A .

Furthermore, M_1 and M_2 agree on all terms and predicates outside of H_E-H_{EB} . (E_0 cannot be a basic event type.) M_2 defeats M_1 's candidacy for minimality in H_E-H_{EB} . Therefore the statement in Theorem 3.1 must hold in all models closed under specialization. Since A-closed models are a subset of those closed under specialization, the statement also holds in all A-closed models.

Q.E.D.

Theorem 3.2

Let EXA be the set of all statements which instantiate Theorem 3.1 for a particular H. If M_1 is a model of $H_A \cup EXA$ such that $:C \in M_1[E_0]$, then there is a basic event type $E_b \in H_{EB}$ such that $:C \in M_1[E_b]$ and E_0 abstracts* E_b .

Proof

By induction. Define a partial ordering over H_E by $E_j < E_k$ iff E_k abstracts E_j . Suppose $E_0 \in H_{EB}$. Then E_0 abstracts* E_0 . Otherwise, suppose the lemma holds for all $E_i < E_0$. Since

$$\forall x . E_0(x) \supset (E_1(x) \vee E_2(x) \vee \dots \vee E_n(x))$$

and $:C \in M_1[E_0]$, it must be the case that

$$:C \in M_1[E_1] \vee \dots \vee :C \in M_1[E_n]$$

WLG, suppose $:C \in M_1[E_1]$. Then there is an E_b such that E_1 abstracts* E_b and $:C \in M_1[E_b]$. Since E_0 abstracts E_1 , it also abstracts* E_b .

Q.E.D.

Theorem 3.3

Every event in a model closed under specialization is of at least one basic type. That is, if M_1 is a model of H_A closed under specialization such that $:C \in M_1[E_0]$, then there is a basic event type $E_b \in H_{EB}$ such that $:C \in M_1[E_b]$ and E_0 abstracts* E_b .

Proof

By Theorem 3.1, $H_A \cup EXA$ holds in M_1 , and by Theorem 3.2 there is such an E_b .

Q.E.D.

Theorem 3.4

M_1 is a model of H_A closed under specialization if and only if M_1 is a model of $H_A \cup EXA$.

Proof

The "only if" half follows from Theorem 3.1. We prove that if M_1 is a model of $H_A \cup EXA$, then M is closed under specialization.

Suppose not. Then there is an M_2 which defeats M_1 . M_2 and M_1 agree on H_{EB} , but there is (at least one) $E_0 \in H_E - H_{EB}$ and event $:C$ such that

$$:C \in M_1[E_0] \wedge :C \notin M_2[E_0]$$

By Theorem 3.2, there is an $E_b \in H_{EB}$ such that

$$:C \in M_1[E_b]$$

Since M_1 and M_2 agree on H_{EB} ,

$$:C \in M_2[E_b]$$

But then because E_0 abstracts E_b

$$:C \in M_2[E_0]$$

which is a contradiction. Therefore there can be no such M_2 . Thus M_1 is closed under specialization.

Q.E.D.

Theorem 3.5 (Disjointedness)

If event predicates E_1 and E_2 are not compatible, then the statement:

$$\forall x. \neg E_1(x) \vee \neg E_2(x)$$

is true in all models of H_A which are closed under abstraction. The statement is also true in all A-closed models of H .

Proof

Suppose M_1 is a model of H_A closed under specialization in which the statement is false. We prove that M_1 is not closed under abstraction.

So let $:C$ be an event such that

$$E_1(x) \wedge E_2(x)$$

is true in $M_1\{x/:C\}$, where E_1 and E_2 are not compatible. Using Theorem 3.3, let E_b be a basic event type abstracted by E_1 such that $:C \in M_1[E_b]$. Define M_2 as follows.

$$\text{Domain}(M_2) = \text{Domain}(M_1)$$

$$M_2[Z] = M_1[Z] \text{ for } Z \in H_E$$

$$M_2[E_i] = M_1[E_i] \text{ if } E_i \text{ abstracts* } E_b$$

$$M_1[E_i] - \{ :C \} \text{ otherwise}$$

In particular, note that $M_1[\text{AnyEvent}] = M_2[\text{AnyEvent}]$, since AnyEvent certainly abstracts E_b . We claim that M_2 is a model of H_A closed under specialization.

(Proof that M_2 is a model of H_A)

Suppose M_2 is not a model of H_A ; in particular, suppose the axiom

$$\forall x. E_j(x) \supset E_i(x)$$

is false in M_2 . Since it is true in M_1 , and M_2 differs from M_1 only in the absence of $:C$ from the extension of some event predicates, it must be the case that

$$:C \in M_2[E_j] \wedge :C \notin M_2[E_i]$$

while

$$:C \in M_1[E_j] \wedge :C \in M_1[E_i]$$

By the definition of M_2 , it must be the case that E_j abstracts* E_b . Since E_i abstracts E_j , then E_i abstracts* E_b as well. But then M_1 and M_2 would have to agree on E_i ; that is,

$$:C \in M_2[E_i]$$

which is a contradiction. So M_2 must be a model of H_A after all.

(Proof that M_2 is closed under specialization)

By Theorem 3.4, M_2 is closed under specialization if it is a model of EXA. So suppose it is not a model of EXA; in particular, suppose

$$\forall x . Ej_0(x) \supset (Ej_1(x) \vee Ej_2(x) \vee \dots \vee Ej_n(x))$$

is false. Then it must be the case that

$$:C \in M_2[Ej_0] \wedge :C \notin M_2[Ej_1] \wedge \dots \wedge :C \notin M_2[Ej_n]$$

But $:C \in M_2[Ej_0]$ means that Ej_0 abstracts* E_b . Since Ej_0 is not basic, at least one of $Ej_1, \dots, Ej_n$ abstracts E_b . WLG, suppose it is Ej_1 . Then

$$:C \in M_2[E_b] \Rightarrow$$

$$:C \in M_2[Ej_1]$$

which is a contradiction. Therefore all members of EXA hold in M_2 , so M_2 is closed under specialization.

(Conclusion of proof)

Furthermore, M_1 and M_2 agree on all terms and predicates outside of $H_E - \{\text{AnyEvent}\}$. Because the extension of E_2 in M_2 is a proper subset of the extension of E_2 in M_1 , M_2 defeats M_1 's candidacy for minimality in $H_E - \{\text{AnyEvent}\}$ among models closed under specialization. Thus the statement in Theorem 3.5 holds in all models closed under abstraction, and the subset of A-closed models.

Q.E.D.

Theorem 3.6

Let DJA be the set of all statements which instantiate Theorem 3.5 for a particular H. If M_1 is a model of $H_A \cup \text{EXA} \cup \text{DJA}$ such that $:C \in M_1[E_0]$, then there is a unique basic event type E_b such that $:C \in M_1[E_b]$. Any event type which holds of $:C$ abstracts* E_b .

Proof

By Theorem 3.2 there must be at least one $E_b \in H_{EB}$ such that $:C \in M_1[E_b]$. By definition any two distinct basic event types are not compatible; thus DJA contains an axiom of the form

$$\forall x. \neg E_b(x) \vee \neg E_{b'}$$

for all other basic event types $E_{b'}$. Therefore E_b is unique. By Theorem 3.2 any event type which holds of $:C$ must abstract* E_b .

Q.E.D.

Theorem 3.7 (Unique Basic Types)

If M_1 is a model of H_A closed under abstraction containing event token $:C_1$, then there is a unique basic event type E_b such that $:C \in M_1[E_b]$. Any event type which holds of $:C$ abstracts* E_b .

Proof

By Theorems 3.1 and 3.5, M_1 is a model of $H_A \cup EXA \cup DJA$. By Theorem 3.6 there is such an E_b .

Q.E.D.

Theorem 3.8

If M_1 is an model of $H_A \cup EXA \cup DJA$, then M_1 is a model of H_A closed under abstraction.

Proof

Suppose not; then there is an M_2 closed under specialization which defeats M_1 for minimality in $H_E - \{\text{AnyEvent}\}$. M_1 and M_2 agree on AnyEvent, but there exists (at least one) $E_0 \in H_E - \{\text{AnyEvent}\}$ and event $:C$ such that

$$:C \in M_1[E_0] \wedge :C \notin M_2[E_0]$$

Now

$$:C \in M_1[E_0] \Rightarrow$$

$$:C \in M_1[\text{AnyEvent}] \Rightarrow$$

$$:C \in M_2[\text{AnyEvent}]$$

By Theorem 3.3 there is some $E_b \in H_{EB}$ such that

$$:C \in M_2[E_b]$$

Since M_2 defeats M_1 ,

$$:C \in M_1[E_b]$$

By Theorem 3.6, E_b is the unique basic type of $:C$ in M_1 , and E_0 abstracts* E_b . But E_0 abstracts* E_b means that

$$:C \in M_2[E_b] \Rightarrow$$

$$:C \in M_2[E_0]$$

which is a contradiction. Therefore there can be no such M_2 , and M_1 must be closed under abstraction.

Q.E.D.

Theorem 3.9 (Abstraction Completeness)

M_1 is an A-closed model of H if and only if M_1 is a model of $H \cup EXA \cup DJA$.

Proof

If M_1 is an A-closed model of H , then it is also a model of H_A closed under abstraction, so by Theorems 3.1 and 3.5 it is a model of $H \cup EXA \cup DJA$. If M_1 is a model of $H \cup EXA \cup DJA$, then it is also a model of $H_A \cup EXA \cup DJA$, so by Theorem 3.8 it is a model of H_A closed under abstraction. Since it is also a model of H , it is an A-closed model of H .

Q.E.D.

Theorem 3.10 (No Useless Events)

Let M_1 be a covering model of H , containing event token $:C_1$. Then either $:C_1 \in M_1[\text{End}]$ is true, or there exists some event token $:C_2$ such that $:C_1$ is a direct component of $:C_2$.

Proof

Suppose the lemma is false: M is a covering model,

$$:C_1 \in M_1[E_1]$$

$$:C_1 \notin M_1[\text{End}]$$

and there does not exist event token which has $:C_1$ as a direct component. Define M_2 as follows.

$$M_2[Z] = M_1[Z] \text{ for } Z \in H_E$$

$$M_2[E] = M_1[E] - \{ :C_1 \} \text{ for } E \in H_E$$

Note that M_1 and M_2 agree on End . If M_2 is an A-closed model of H_A , then M_2 defeats M_1 for minimality in $H_E - \{\text{End}\}$. We will prove this by showing that $H \cup \text{EXA} \cup \text{DJA}$ holds in M_2 . We consider each of the types of axioms in turn.

(Case 1) Axioms in H_G must hold, because they receive the same valuation in M_1 and M_2 .

(Case 2) Axioms in H_A are of the form:

$$\forall x . E_j(x) \supset E_i(x)$$

Suppose one is false; then for some $:D$,

$$:D \in M_2[E_j] \wedge :D \notin M_2[E_i]$$

But this is impossible, because M_1 and M_2 must agree when $:D \neq :C_1$, as must be case, because $:C_1$ does not appear in the extension of any event type in M_2 .

(Case 3) Axioms in EXA must hold by the same argument.

(Case 4) Axioms in DJA must hold because they contain no positive uses of H_E .

(Case 5) The j -th axiom in H_D is of the form:

$\forall x . E_{j0}(x) \supset E_{j1}(f_{j1}(x)) \wedge E_{j2}(f_{j2}(x)) \wedge \dots \wedge E_{jn}(f_{jn}(x)) \wedge \kappa$
 Suppose it does not hold in M_2 . Then there must be some $:C_2$ such that

$$E_{j0}(x) \wedge \{ \neg E_{j1}(f_{j1}(x)) \vee \\ \neg E_{j2}(f_{j2}(x)) \vee \dots \vee \\ \neg E_{jn}(f_{jn}(x)) \vee \\ \neg \kappa \}$$

is true in $M_2\{x/:C_2\}$. M_1 and M_2 agree on κ , so it must be the case that for some j_i ,

$$M_2[f_{ji}](:C_2) \in M_2[E_{ji}]$$

while

$$M_1[f_{ji}](:C_2) \in M_1[E_{ji}]$$

Because M_1 and M_2 differ on E_{ji} only at $:C_1$, it must be the case that

$$M_1[f_{ji}](:C_2) = :C_1$$

But then $:C_1$ is a component of $:C_2$ in M_1 , contrary to our original assumption.

(End of Cases)

By Theorem 3.9 M_2 is A-closed, and so it defeats M_1 . This contradiction proves the theorem.

Q.E.D.

Theorem 3.11 (Component/Use)

Let $E \in H_E$, and $\text{Com}(E)$ be the set of event predicates with which E is compatible.

Consider all the decomposition axioms in which any element of $\text{Com}(E)$ appears on the right-hand side. The j -th such decomposition axiom has the following form, where E_{ji} is the element of $\text{Com}(E)$:

$$\forall x . E_{j0}(x) \supset E_{j1}(f_{j1}(x)) \wedge \dots \wedge E_{ji}(f_{ji}(x)) \wedge \dots \wedge E_{jn}(f_{jn}(x)) \wedge \kappa$$

Suppose that the series of these axioms, where an axiom is repeated as many times as there are members of $\text{Com}(E)$ in its right-hand side, is of length $m > 0$. Then the following statement is c-valid:

$$\begin{aligned}
\forall x . E(x) \supset & \text{End}(x) \vee \\
& (\exists y . E_{1,0}(y) \wedge f_{1i}(y)=x) \vee \\
& (\exists y . E_{2,0}(y) \wedge f_{2i}(y)=x) \vee \\
& \dots \vee \\
& (\exists y . E_{m,0}(y) \wedge f_{mi}(y)=x)
\end{aligned}$$

Proof

Let M be a covering model such that $:C \in M[E]$ and $:C \in M[\text{End}]$. By Theorem 3.10 there is an E_{j0} , E_{ji} , and $:D$ such that

$$:D \in M[E_{j0}]$$

$$:C \in M[E_{ji}]$$

$$:C = M[f_{ji}](:D)$$

where f_{ji} is a role function in a decomposition axiom for E_{j0} . By Theorem 3.5, E and E_{ji} are compatible. By inspection we see that the second half of the formula above is true in M when x is bound to $:C$, because the disjunct containing E_{ji} is true when the variable y is bound to $:D$. Since the choice of $:C$ was arbitrary, the entire formula is true in M . Finally, since M could be any covering model, the formula is c-valid.

Q.E.D.

Theorem 3.12

Let CUA be the set of all formulas which instantiate Theorem 3.11 for a particular H . If M_1 is a model of $H \cup EXA \cup DJA \cup CUA$ such that $:C \in M_1[E]$, then there is a $:C_n$ such that $:C_n \in M_1[\text{End}]$ and $:C$ is a component* of $:C_n$.

Proof

If $:C_1 \in M_1[\text{End}]$, then $:C_1$ is a component* of $:C_n = :C_1$. So suppose $:C_1 \notin M_1[\text{End}]$. For the axioms in CUA to hold there must be sequences of event tokens, types, and role functions of the following form:

$:C_1$	$:C_1$	$:C_3$	$:C_3$	$:C_5$	$:C_5$	...
E_1	E_2	E_3	E_4	E_5	E_6	...
		f_{3i}		f_{5i}		...

such that

For all j , $:C_j \in M_1[E_j]$

For odd j , E_j and E_{j+1} are compatible

For odd j , $j \geq 3$, E_{j-1} is a direct component of E_j , and $:C_{j-2} = M_1[f_{ji}](:C_j)$

This sequence must terminate with a $:C_n$ such that $:C_n \in M_1[End]$. Otherwise, some E_i must appear more than once in the sequence, which would mean that a cycle exists in H ; but we have supposed that H is acyclic. Therefore $:C_1$ is a component* of C_n .

Q.E.D.

Theorem 3.13 (No Infinite Chains)

If M_1 is a covering model of H such that $C_1 \in M_1[E]$, then there is a $:C_n$ such that $:C_n \in M_1[End]$ and $:C_1$ is a component* of $:C_n$.

Proof

By Theorems 3.9 and 3.8 M_1 is a model of $H \cup EXA \cup DJA \cup CUA$; by Theorem 3.12 there is such a $:C_n$.

Q.E.D.

Theorem 3.14 (Decomposition Completeness)

M_1 is a covering model of H if and only if M_1 is a model of $H \cup EXA \cup DJA \cup CUA$.

Proof

The "only if" half follows from Theorems 3.9 and 3.11. We prove that if M_1 is a model of $H \cup EXA \cup DJA \cup CUA$, then it is a covering model.

Suppose not. By Theorem 3.9, M_1 is A-closed, so there must be an A-closed M_2 which defeats M_1 for minimality in $H_E - \{End\}$. M_1 and M_2 agree on all functions and predicates outside of $H_E - \{End\}$, but there exists (at least one) $E_1 \in H_E - \{End\}$ and event $:C_1$ such that

$$:C_1 \in M_1[E_1]$$

$$:C_1 \notin M_2[E_1]$$

By Theorem 3.12 there is a $:C_n$ such that

$$:C_n \in M_1[End]$$

and $:C_1$ is a component* of $:C_n$. By the construction used in the proof of Theorem 3.12 there are sequences

$$\begin{array}{ccccccc} :C_1 & :C_1 & :C_3 & :C_3 & \dots & :C_n & :C_n \\ E_1 & E_2 & E_3 & E_4 & \dots & E_n & E_{n+1} = End \\ & & f_{3i} & & \dots & f_{ni} & \end{array}$$

such that

$$\text{For all } j, 1 \leq j \leq n+1, :C_j \in M_1[E_j]$$

$$\text{For odd } j, E_j \text{ and } E_{j+1} \text{ are compatible}$$

$$\text{For odd } j, 3 \leq j \leq n$$

$$E_{j-1} \text{ is a direct component of } E_j$$

$$:C_{j-2} = M_1[f_{ji}](:C_j) = M_2[f_{ji}](:C_j)$$

Because M_1 and M_2 agree on End,

$$:C_n \in M_2[End]$$

Now for any odd $j, 3 \leq j \leq n$, if

$$:C_j \in M_2[E_j]$$

then since H_D holds in M_2 ,

$$:C_{j-2} \in M_2[E_{j-1}]$$

If we prove that for all odd $j, 1 \leq j \leq n$

$$:C_j \in M_2[E_{j+1}] \Rightarrow :C_j \in M_2[E_j]$$

we will be done; because we would then know that

$$\begin{aligned}
& :C_n \in M_2[\text{End}] \Rightarrow \\
& :C_n \in M_2[E_{n+1}] \Rightarrow \\
& :C_n \in M_2[E_n] \Rightarrow \\
& :C_{n-2} \in M_2[E_{n-1}] \Rightarrow \\
& \dots \Rightarrow \\
& :C_3 \in M_2[E_3] \Rightarrow \\
& :C_1 \in M_2[E_2] \Rightarrow \\
& :C_1 \in M_2[E_1] \Rightarrow
\end{aligned}$$

which yields the desired contradiction. So let's prove that

$$:C_j \in M_2[E_{j+1}] \Rightarrow :C_j \in M_2[E_j]$$

Assume the antecedent

$$:C_j \in M_2[E_{j+1}]$$

Because M_2 is A-closed, there is a unique $E_b \in H_{EB}$ such that

$$:C_j \in M_2[E_b]$$

Because M_2 defeats M_1 ,

$$:C_j \in M_1[E_b]$$

Since M_1 is A-closed, Theorem 3.7 tells us that because

$$:C_j \in M_1[E_j]$$

it must be the case that E_j abstracts* E_b . But then since H_A holds in M_2 ,

$$:C_j \in M_2[E_j]$$

and we are done.

Q.E.D.

Theorem 3.15 (Computability of C-Entailment)

There is a computable function cl which maps a hierarchy H into a set of axioms with the property that

$$Q_H \models_c P$$

if and only if

$$cl(H) \cup Q \models P$$

Proof

The function simply computes $H \cup EXA \cup DJA \cup CUA$, which is a finite, recursively enumerable set. Theorem 3.14 guarantees the equivalence.

Theorem 3.16 (Not Predicate Completion)

Theorem 3.11 cannot be strengthened by considering only axioms in which E appears as a component, instead of all axioms in which event types compatible with E appear as components.

Proof

Consider the following hierarchy:

H_A

- $\forall x . E_1(x) \supset \text{AnyEvent}(x)$
- $\forall x . E_2(x) \supset \text{AnyEvent}(x)$
- $\forall x . E_3(x) \supset E_1(x)$
- $\forall x . E_3(x) \supset E_2(x)$
- $\forall x . \text{End}(x) \supset \text{AnyEvent}(x)$
- $\forall x . E_4(x) \supset \text{End}(x)$
- $\forall x . E_5(x) \supset \text{End}(x)$

H_D

- $\forall x . E_4(x) \supset E_1(f_4(x)) \wedge P$
- $\forall x . E_5(x) \supset E_2(f_5(x)) \wedge Q$

The modified statement would not hold in the covering model

$\{ \text{AnyEvent}(D), \text{End}(D), E_4(D), E_1(f_4(D)), \\ E_3(f_4(D)), E_2(f_4(D)), \text{AnyEvent}(f_4(D)) \}$

because even though $E_1(f_4(D))$ is true, there is no instance of E_5 .

Q.E.D.

Theorem 3.17 (C-entailment and Circumscription)

For a given a hierarchy H , a statement P is c-entailed by Q if and only if P follows from the following schema:

$$Q \wedge \text{Circum}(H \wedge \text{Circum}(\text{Circum}(H_A, H_E - H_{EB}), \\ \{\text{AnyEvent}\}), \\ H_E - \{\text{End}\})$$

Note that the two inner circumscriptions apply only to the abstraction hierarchy, while the final circumscription applies to all of H .

Proof

See [Etherington 86] for proof of the correctness of circumscription. Since the result of all the minimizations can be described by a finite set of axioms, we strongly suspect that circumscription is complete in this case. (No proof, however, has appeared in the literature.)

Q.E.D.

Appendix B

Chapter 4 Proofs

Theorem 4.1 (Minimum Cardinality Defaults)

Consider the following sequences of statements.

$$MA_0. \quad \forall x. \neg \text{End}(x)$$

$$MA_1. \quad \forall x, y. \text{End}(x) \wedge \text{End}(y) \supset x=y$$

$$MA_2. \quad \forall x, y, z. \text{End}(x) \wedge \text{End}(y) \wedge \text{End}(z)$$

$$\supset (x=y) \vee (x=z) \vee (y=z)$$

...

The first asserts that no End events exist; the second, no more than one End event exists; the third, no more than two; and so on. Let P be any sentence, Γ any sentence or set of sentences, and H a hierarchy. Suppose there is a minimum covering model in which the extension of End is finite. Then

$$\Gamma \cup H \models_{mc} \Omega$$

if and only if

$$\Gamma \cup \text{cl}(H) \cup MA_i \vdash \Omega$$

where i is the smallest integer such that left-hand side of the provability relation is consistent.

Proof

We'll prove that M_1 is a minimum cover of Γ relative to H (with finite extension of End) if and only if M_1 is a model of $\Gamma \cup \text{cl}(H) \cup MA_i$. The theorem then follows from completeness and correctness of first-order logic.

Suppose M_1 is a model of $\Gamma \cup \text{cl}(H) \cup MA_i$. By Theorem 3.14, M_1 is a covering model. Clearly $|M_1[\text{End}]| \leq i$. Suppose that M_1 were defeated for minimum cardinality in End by M_2 . Then $|M_2[\text{End}]| = j < i$. But by Theorem 3.14 M_2 is a model of $\Gamma \cup \text{cl}(H)$. Because $|M_2[\text{End}]| = j$, M_2 is also a model of MA_j . But this is

impossible, because we've assumed that $\Gamma \cup \text{cl}(H) \cup \text{MA}_j$ is inconsistent. So M_1 cannot be defeated, and must be a minimum cover.

Suppose M_1 is such a minimum cover, where $|M_1[\text{End}]| = n$. As before, M_1 must be model of $\Gamma \cup \text{cl}(H)$. Suppose it were not a model of MA_i . Then $|M_1[\text{End}]| > i$. But because $\Gamma \cup \text{cl}(H) \cup \text{MA}_i$ is consistent, it has a model M_2 , which is also be a covering model, with $|M_2[\text{End}]| \leq i$. But this is impossible, because M_2 would defeat M_1 . Therefore MA_i must also be true in M_1 .

Q.E.D.

Theorem 4.2 (Cardinality Circumscription)

Let α include all the predicate, function, and constant symbols in our language other than End. Suppose that all models of H are infinite, and in some model of $\Gamma \cup \text{cl}(H)$, End has a finite extension. If

$$\text{Circum}(\Gamma \cup \text{cl}(H), \{\text{End}\}, \alpha) \vdash \Omega$$

then

$$\Gamma_H \models_{\text{mc}} \Omega$$

where $\text{Circum}(\Gamma \cup \text{cl}(H), \{\text{End}\}, \alpha)$ means to circumscribe with α varying. The "if" is strengthened to "if and only if" if it is true that circumscription is complete in this case.

Proof

We prove that M_1 is a minimum covering model of Γ relative to H if and only if M_1 is minimal in End among models of $\Gamma \cup \text{cl}(H)$ where α varies. The theorem then follows from correctness of circumscription. The strengthened version of the theorem depends upon the truth of the proposition that circumscription is complete when the result of the circumscription is equivalent to a finite set of first-order formulas. Although this proposition is very likely to be true, no proof has yet been discovered [Etherington 87].

(if) Suppose M_1 is a minimum covering model of Γ . By Theorem 3.14 M_1 is a model of $\Gamma \cup \text{cl}(H)$. Suppose M_1 were *not* minimal in End where α varies; that is, there is an M_2 such that

1. M_2 is a model of $\Gamma \cup \text{cl}(H)$
2. $\text{domain}(M_1) = \text{domain}(M_2)$
3. $M_2[\text{End}] \subset M_1[\text{End}]$

But (1) implies M_2 is a covering model, and (2) implies that $|M_2[\text{End}]| < |M_1[\text{End}]|$, so M_2 would defeat M_1 's candidacy as a minimum covering model. So there can be no such M_2 , and M_1 is minimal in End where α varies.

(only if) Suppose M_1 is minimal in End among models of $\Gamma \cup \text{cl}(H)$ where α varies. Suppose M_1 is *not* a minimum covering model of Γ . Since M_1 is a covering model, there must be an M_2 which defeats M_1 's candidacy to be a minimum covering model, with

$$|M_2[\text{End}]| < |M_1[\text{End}]|$$

We will construct a model which defeats M_1 's candidacy for minimality in End where α varies, yielding the desired contradiction. But M_2 itself is not suitable, because its domain and M_1 's domain may differ, and may not even be of the same size. So consider the following two cases:

1. Suppose $|\text{Domain}(M_1)| \leq |\text{Domain}(M_2)|$. Apply the Downward Tarski-Löwenheim-Skolem Theorem [Barwise 77] and let M_3 be an elementary submodel of M_2 such that $|\text{Domain}(M_1)| = |\text{Domain}(M_3)|$.

2. Otherwise $|\text{Domain}(M_1)| > |\text{Domain}(M_2)|$. Apply the Upward Tarski-Löwenheim-Skolem Theorem and let M_3 be an elementary extension of M_2 such that $|\text{Domain}(M_1)| = |\text{Domain}(M_3)|$.

Because M_3 is an elementary submodel or extension, and M_2 is a model of $\Gamma \cup \text{cl}(H)$, M_3 is a model of $\Gamma \cup \text{cl}(H)$. Furthermore, we have assumed that $|M_2[\text{End}]|$ is finite, let's say of size n . Then the sentence MA_n is true in M_2 , and therefore also in M_3 . This means that $|M_3[\text{End}]| \leq |M_2[\text{End}]|$. Thus M_3 is a covering model which defeats M_1 's candidacy to be a minimum covering model.

Let M_4 be a homomorphism of M_3 such that $\text{Domain}(M_4) = \text{Domain}(M_1)$. Now we have a model of $\Gamma \cup \text{cl}(H)$ which has the same domain as M_1 , but is smaller in the size of its extension of End . But we need a model where the extension of End which is a subset of that in M_1 .

Suppose

$$M_1[\text{End}] = \{ :A_1, :A_2, \dots :A_j, :C_1, :C_2, \dots :C_k, :C_{k+1}, \dots \}$$

$$M_4[\text{End}] = \{ :A_1, :A_2, \dots :A_j, :B_1, :B_2, \dots :B_k \}$$

That is, $:A_1$ through $:A_j$ (where possibly $j=0$) are the elements in End that M_1 and M_2 have in common. Elements $:C_1$ and up are in End only in M_1 , and $:B_1$ through $:B_k$ are only in End in M_4 . The postfix substitution operator $\{ :A/:B \}$ replaces all instances of $:A$ by instances of $:B$. Let θ be the substitution which swaps all the $:B_i$ with $:C_i$.

$$\theta = \{ :B_1/:C_1, \dots :B_k/:C_k, :C_1/:B_1, \dots :C_k/:B_k \}$$

Define M_5 as follows.

$$M_5[p] = M_2[p]\theta \text{ for all predicates } p$$

$$M_5[c] = M_2[c]\theta \text{ for all constants } c$$

$$M_5[f] = \lambda x_1, x_2, \dots x_n . (M_2[f](x_1 \theta, x_2 \theta, \dots x_n \theta))\theta$$

for all n -ary functions f

M_5 is a homomorphism of M_4 , and is therefore a model of $\Gamma \cup \text{cl}(H)$. The proper subset condition is satisfied because

$$M_5[\text{End}] = \{ :A_1, \dots :A_j, :C_1, \dots :C_k \} \subset M_1[\text{End}]$$

Because α includes all symbols other than End, M_1 and M_5 do not have to agree on the interpretation of any symbols. So M_5 defeats M_1 's candidacy for minimality in End where α varies, which is a contradiction. Therefore M_1 is a minimum covering model of Γ .

Q.E.D.

Appendix C

Chapter 5 Proofs

Theorem 5.1 (Non-Universal Conclusions)

Let Ω be a sentence which, when written with all quantifiers in initial position, contains no universal quantifiers. Then

$$\Gamma_H \models_{\text{mcs}} \Omega$$

if and only if

$$\Gamma_H \models_{\text{mc}} \Omega$$

Proof

(Only if) Trivial, since a minimum cover is an mcs-model.

(If) Ω can be written

$$\exists x_1, \dots, x_n. \omega$$

where ω is a sentence containing no quantifiers in which $x_1, \dots, x_n$ appear free. Suppose Ω holds in all minimum covers. Let M_1 be any mcs-model. Then M_1 contains submodel M_2 in which

$$\exists x_1, \dots, x_n. \omega$$

But this is only the case if there are

$$:C_1, \dots, :C_n \in \text{Domain}(M_2)$$

such that ω is true in $M_2\{x_1/:C_1, \dots, x_n/:C_n\}$. A straightforward inductive argument shows that ω is true in $M_1\{x_1/:C_1, \dots, x_n/:C_n\}$ because all terms in ω are interpreted by $M_1\{x_1/:C_1, \dots, x_n/:C_n\}$ as individuals in the domain of M_2 . Therefore Ω holds in M_1 , and in all mcs-models.

Q.E.D.

Theorem 5.2 (Incremental Recognition)

In the case of a single observation, imc-entailment is the same as mcs-entailment.

$$(\Gamma_1)_H \models_{\text{imc}} \Omega$$

if and only if

$$\Gamma_1_H \models_{\text{mcs}} \Omega$$

In the case of multiple observations, imc-entailment is monotonic.

$$(\Gamma_1, \dots, \Gamma_{n-1})_H \models_{\text{imc}} \Omega$$

implies

$$(\Gamma_1, \dots, \Gamma_{n-1}, \Gamma_n)_H \models_{\text{imc}} \Omega$$

Proof

The first part of the theorem follows from the base case of the definition of imc-entailment. The second part of the theorem follows from the fact that an incremental minimum cover of $(\Gamma_1, \dots, \Gamma_{n-1}, \Gamma_n)$ must also be an incremental minimum cover of $(\Gamma_1, \dots, \Gamma_{n-1})$. Therefore if Ω holds in all models of the latter sort, it must also hold in all models of the former sort.

Appendix D

Temporal Constraint Logic

The following rules translate Allen's temporal interval operators into an algebra on fuzzy interval constraints. A *time interval* is represented by a term which is *interpreted* as a pair of real numbers. The functions *first-instant* and *last-instant* apply to a interval and yield the lower and upper bounds respectively. A *fuzzy interval constraint* is a tuple of four real numbers. A time interval *satisfies* a fuzzy constraint, written

$$I \in (\text{start-min}, \text{start-max}, \text{end-min}, \text{end-max})$$

if the first instance of I falls between start-min and start-max inclusive, and the final instance of I falls between end-min and end-max inclusion. That is,

$$(\text{start-min} \leq \text{first-instant}(I) \leq \text{start-max}) \wedge \\ (\text{end-min} \leq \text{last-instant}(I) \leq \text{end-max})$$

While start-min, start-max, etc., are represented by *numerals*, first-instant(I) and last-instant(I) remain *symbolic* quantities.

The form of each each rule is as follows: given that T_1 satisfies fuzzy constraint Z_1 , and T_2 satisfies fuzzy constraint Z_2 , and T_1 is related by a given binary-operator to T_2 , then it must be the case that T_1 satisfies fuzzy constraint Z_3 . The logical form of the rules is as follows, where $Z_3 = \text{Ftransitive}(Z_1, \text{binary-op}, Z_2)$.

Where T_1, T_2 are time intervals, and Z_1, Z_2 are fuzzy constraints:

$$T_1 \in Z_1 \wedge T_2 \in Z_2 \wedge T_1 \text{ binary-op } T_2 \supset \\ T_1 \in \text{Ftransitive}(Z_1, \text{binary-op}, Z_2)$$

Suppose that T_1 and T_2 are related by a *disjunction* of temporal operators. Then one calculates Z_3 for each alternative, and then combine all the answers with the *Funion* function, which is defined below. In formal terms:

$$T_1 \in Z_1 \wedge T_2 \in Z_2 \wedge T_1 (\text{op}_1 \text{ op}_2 \dots \text{op}_n) T_2 \supset \\ T_1 \in \text{Funion}(\text{Ftransitive}(Z_1, \text{op}_1, Z_2), \\ \text{Ftransitive}(Z_1, \text{op}_2, Z_2), \\ \dots, \text{Ftransitive}(Z_1, \text{op}_n, Z_2))$$

$$\text{Where } T_1 (\text{op}_1 \text{op}_2 \dots \text{op}_n) T_2 \equiv \\ (T_1 \text{op}_1 T_2) \vee (T_1 \text{op}_2 T_2) \vee \dots \vee (T_1 \text{op}_n T_2)$$

Consider the case where an interval is known to satisfy two different fuzzy constraints. Then it must be the case that the interval satisfies the Fintersection of the constraints. Formally:

$$T_1 \in Z_1 \wedge T_1 \in Z_2 \supset T_1 \in \text{Fintersection}(Z_1, Z_2)$$

Finally, the predicate empty is true of a fuzzy constraint just in case no interval could satisfy it.

The following tables define the Funion, Fintersection, empty and Ftransitive functions. Their correctness can be verified by elementary algebra.

$$\text{Funion}((a_1 \ b_1 \ c_1 \ d_1), \dots, (a_n \ b_n \ c_n \ d_n)) = \\ (\min(a_1, \dots, a_n) \ \max(b_1, \dots, b_n) \ \min(c_1, \dots, c_n) \ \max(d_1, \dots, d_n))$$

$$\text{Fintersection}((a_1 \ b_1 \ c_1 \ d_1), \dots, (a_n \ b_n \ c_n \ d_n)) = \\ (\max(a_1, \dots, a_n) \ \min(b_1, \dots, b_n) \ \max(c_1, \dots, c_n) \ \min(d_1, \dots, d_n))$$

$$\text{empty}((a \ b \ c \ d)) = \\ a > b \vee c > d \vee a > d$$

Ftransitive((a b c d), op, (e f g h)) = (w x y z)				
op	w	x	y	z
before	a	min(b,f)	c	min(d,f)
meets	a	min(b,f)	max(c,e)	min(d,f)
overlaps	a	min(b,f)	max(c,e)	min(d,h)
starts	max(a,e)	min(b,f)	max(c,e)	min(d,h)
during	max(a,e)	min(b,h)	max(c,e)	min(d,h)
finishes	max(a,e)	min(b,h)	max(c,g)	min(d,h)
overlapped by	max(a,e)	min(b,h)	max(c,g)	d
met by	max(a,g)	min(b,h)	max(c,g)	d
after	max(a,g)	b	max(c,g)	d

Appendix E

Transcripts

Following are transcripts of the implementation running on some of the examples discussed in this thesis.

Hunt/Rob Example

:: First observation: get-gun(GET-GUN1). Builds graph named g1.

Command: (EXPLAIN-OBSERVATION '(GET-GUN1 GET-GUN) 'G1)

T

:: Display e-graph g1.

:: There are two alternatives, with GET-GUN1 as component of rob-bank or of hunt.

Command: (DRAW-GRAPH)

graph G1

END226

?= HUNT228

?= HUNT227

S1 -> GET-GUN1

?= ROB-BANK225

?= ROB-BANK224

S1 -> GET-GUN1 ^

*The e-graphs are printed out linearly, where the symbol
?= represents an alternative link and rolenam->
represents a component (and) link. The symbol ^
indicates a link to structure already printed above. This
description is equivalent to the following diagram:*

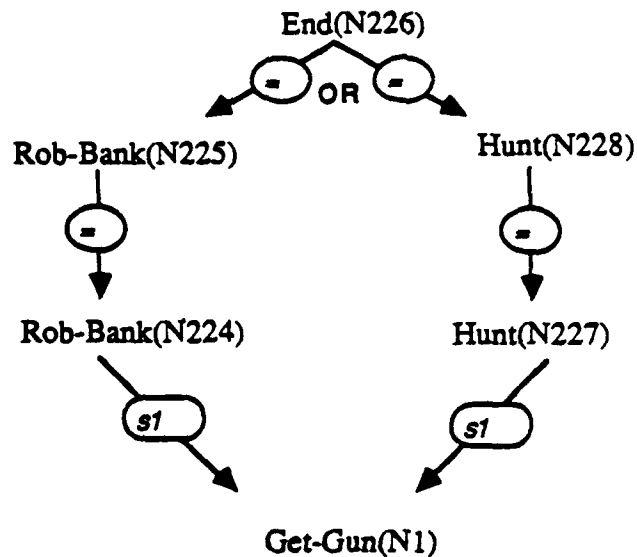


figure E.1: E-Graph for Get-Gun

The one difference from the algorithm described in Chapter 7 is the inclusion of nodes which "abstract away" the steps of any node which has explicit component links. In this example, HUNT228 abstracts HUNT227, but doesn't include the component S1. These extra nodes do not change the semantics of e-graphs, but they do allow a more compact representation of the graphs when the type to be explained can fill several different roles in a user type. In such a case, a different user node is created for each use, but all can become alternatives for the same abstract node of that user type.

:: Turn on tracing.

Command: (SETQ *NOISY* T)

T

:: Second observation: go-to-bank(GO-TO-BANK2), in graph G2.

Command: (EXPLAIN-OBSERVATION '(GO-TO-BANK2 GO-TO-BANK) 'G2)

... creating node GO-TO-BANK2

... searching up from GO-TO-BANK2

... creating node CASH-CHECK229
 ... considering GO-TO-BANK2 as S1 of CASH-CHECK229
 ... checking constraints on CASH-CHECK229
 ... CASH-CHECK229 is okay
 ... searching up from CASH-CHECK229
 ... creating node CASH-CHECK230
 ... searching up from CASH-CHECK230
 ... creating node END231
 ... searching up from END231
 ... creating node ROB-BANK232
 ... considering GO-TO-BANK2 as S2 of ROB-BANK232
 ... checking constraints on ROB-BANK232
 ... ROB-BANK232 is okay
 ... searching up from ROB-BANK232
 ... creating node ROB-BANK233
 ... searching up from ROB-BANK233
 ... merging graph at END231

T

:: Show g2, again two alternatives: rob-bank or cash-check.

Command: (DRAW-GRAPH)

graph G2

END231

?= ROB-BANK233

?= ROB-BANK232

S2 -> GO-TO-BANK2

?= CASH-CHECK230

?= CASH-CHECK229

S1 -> GO-TO-BANK2 ^

:: Matching the graphs yields just the rob-bank alternative.

Command: (MATCH-GRAPHS 'G1 'G2 'G1+G2)

... trying to match END226 with END231
... creating node END234
... checking constraints on END234
... END234 is okay
... checking constraints on END234
... END234 is okay
... trying to match HUNT228 with ROB-BANK233
... trying to match HUNT228 with CASH-CHECK230
... trying to match ROB-BANK225 with ROB-BANK233
... creating node ROB-BANK235
... checking constraints on ROB-BANK235
... ROB-BANK235 is okay
... checking constraints on ROB-BANK235
... ROB-BANK235 is okay
... trying to match ROB-BANK224 with ROB-BANK232
... creating node ROB-BANK236
... checking constraints on ROB-BANK236
... ROB-BANK236 is okay
... creating node GO-TO-BANK237
... checking constraints on GO-TO-BANK237
... GO-TO-BANK237 is okay
... checking constraints on GO-TO-BANK237
... GO-TO-BANK237 is okay
... copying GO-TO-BANK2 yields GO-TO-BANK237
... creating node GET-GUN238
... checking constraints on GET-GUN238
... GET-GUN238 is okay
... checking constraints on GET-GUN238
... GET-GUN238 is okay
... copying C1 yields GET-GUN238
... checking constraints on ROB-BANK236
... ROB-BANK236 is okay
... successful match of ROB-BANK224 and ROB-BANK232 yields
ROB-BANK236

... successful match of ROB-BANK225 and ROB-BANK233 yields
ROB-BANK235
... trying to match ROB-BANK225 with CASH-CHECK230
... successful match of END226 and END231 yields END234
END234

Command: (DRAW-GRAPH)

graph G1+G2

END234

?= ROB-BANK235

?= ROB-BANK236

S1 -> GET-GUN238

S2 -> GO-TO-BANK237

Cooking Examples

:: First observation is make-sauce(OBS-SAUCE2), with agent Joe, during
:: time interval beginning between 4 and 5, and ending between 6 and 7.

*Note that fuzzy time constraints are represented by
VECTORS with TIME as first element. The parameters
of the observation are specified by a list of (role value)
pairs. The first use of the symbol "time" below refers to
the role Time, while the second merely identifies the
vector value as a time constraint.*

Command: (EXPLAIN-OBSERVATION '(OBS-SAUCE2 MAKE-SAUCE
(AGENT JOE) (TIME #(TIME 4 5 6 7))))

... creating node OBS-SAUCE2

... searching up from OBS-SAUCE2

... creating node MAKE-PASTA-DISH246

... considering OBS-SAUCE2 as S2 of MAKE-PASTA-DISH246

... checking constraints on MAKE-PASTA-DISH246

... MAKE-PASTA-DISH246 is okay
 ... searching up from MAKE-PASTA-DISH246
 ... creating node MAKE-PASTA-DISH247
 ... searching up from MAKE-PASTA-DISH247
 ... creating node PREPARE-MEAL248
 ... searching up from PREPARE-MEAL248
 ... creating node END249
 ... searching up from END249
 ... creating node MAKE-MARINARA250
 ... checking constraints on MAKE-MARINARA250
 ... MAKE-MARINARA250 is okay
 ... searching up from MAKE-MARINARA250
 ... creating node MAKE-CHICKEN-MARINARA251
 ... considering MAKE-MARINARA250 as S5 of
 MAKE-CHICKEN-MARINARA251
 ... checking constraints on MAKE-CHICKEN-MARINARA251
 ... MAKE-CHICKEN-MARINARA251 is okay
 ... searching up from MAKE-CHICKEN-MARINARA251
 ... creating node MAKE-CHICKEN-MARINARA252
 ... searching up from MAKE-CHICKEN-MARINARA252
 ... creating node MAKE-MEAT-DISH253
 ... searching up from MAKE-MEAT-DISH253
 ... merging graph at PREPARE-MEAL248
 T

:: View the graph.

The parameters of each node appear in a list of role/value pairs after the node name. Only parameters for which a value has been calculated are included.

Command: (DRAW-GRAPH T)

graph G2

END249 NIL

?= PREPARE-MEAL248 ((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

?= MAKE-MEAT-DISH253

((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

?= MAKE-CHICKEN-MARINARA252

((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

?= MAKE-CHICKEN-MARINARA251

((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

S5 -> MAKE-MARINARA250

((AGENT JOE) (TIME #(TIME 4 5 6 7)))

?= MAKE-PASTA-DISH247

((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

?= MAKE-PASTA-DISH246

((AGENT JOE) (TIME #(TIME -INF 5 6 +INF)))

S2 -> OBS-SAUCE2 ((AGENT JOE) (TIME #(TIME 4 5 6 7)))

:: The second observation is making noodles.

Command: (EXPLAIN-OBSERVATION '(OBS-NOODLES3 MAKE-NOODLES
(AGENT JOE) #(TIME 6 8 7 9)))

T

Command: (DRAW-GRAPH T)

graph G3

END257 NIL

?= PREPARE-MEAL256

((AGENT JOE) (TIME #(TIME -INF 8 7 +INF)))

?= MAKE-PASTA-DISH255

((AGENT JOE) (TIME #(TIME -INF 8 7 +INF)))

?= MAKE-PASTA-DISH254

((AGENT JOE) (TIME #(TIME -INF 8 7 +INF)))

S1 -> OBS-NOODLES3 ((AGENT JOE) (TIME #(TIME 6 8 7 9)))

:: We can merge G2 and G3 together: they can be steps of the same action.

Command: (MATCH-GRAPHS 'G2 'G3 'G2+G3)

- ... trying to match END249 with END257
- ... creating node END258
- ... checking constraints on END258
- ... END258 is okay
- ... checking constraints on END258
- ... END258 is okay
- ... trying to match PREPARE-MEAL248 with PREPARE-MEAL256
- ... creating node PREPARE-MEAL259
- ... checking constraints on PREPARE-MEAL259
- ... PREPARE-MEAL259 is okay
- ... checking constraints on PREPARE-MEAL259
- ... PREPARE-MEAL259 is okay
- ... trying to match MAKE-MEAT-DISH253 with MAKE-PASTA-DISH255
- ... trying to match MAKE-PASTA-DISH247 with MAKE-PASTA-DISH255
- ... creating node MAKE-PASTA-DISH260
- ... checking constraints on MAKE-PASTA-DISH260
- ... MAKE-PASTA-DISH260 is okay
- ... checking constraints on MAKE-PASTA-DISH260
- ... MAKE-PASTA-DISH260 is okay
- ... trying to match MAKE-PASTA-DISH246 with MAKE-PASTA-DISH254
- ... creating node MAKE-PASTA-DISH261
- ... checking constraints on MAKE-PASTA-DISH261
- ... MAKE-PASTA-DISH261 is okay
- ... creating node MAKE-NOODLES262
- ... checking constraints on MAKE-NOODLES262
- ... MAKE-NOODLES262 is okay
- ... checking constraints on MAKE-NOODLES262
- ... MAKE-NOODLES262 is okay
- ... copying OBS-NOODLES3 yields MAKE-NOODLES262
- ... creating node MAKE-SAUCE263
- ... checking constraints on MAKE-SAUCE263
- ... MAKE-SAUCE263 is okay
- ... checking constraints on MAKE-SAUCE263

... MAKE-SAUCE263 is okay
 ... copying OBS-SAUCE2 yields MAKE-SAUCE263
 ... checking constraints on MAKE-PASTA-DISH261
 ... MAKE-PASTA-DISH261 is okay
 ... successful match of MAKE-PASTA-DISH246 and MAKE-PASTA-DISH254
 yields MAKE-PASTA-DISH260
 ... successful match of MAKE-PASTA-DISH247 and MAKE-PASTA-DISH255
 yields MAKE-PASTA-DISH261
 ... successful match of PREPARE-MEAL248 and PREPARE-MEAL256 yields
 PREPARE-MEAL259
 ... successful match of END249 and END257 yields END258
 END258

Command: (DRAW-GRAPH T)

graph G2+G3

END258 NIL

?= PREPARE-MEAL259 ((AGENT JOE) (TIME #(TIME -INF 5 7 +INF)))

?= MAKE-PASTA-DISH260

((AGENT JOE) (TIME #(TIME -INF 5 7 +INF)))

?= MAKE-PASTA-DISH261

((AGENT JOE) (TIME #(TIME -INF 5 7 +INF)))

S2 -> MAKE-SAUCE263

((AGENT JOE) (TIME #(TIME 4 5 6 7)))

S1 -> MAKE-NOODLES262

((AGENT JOE) (TIME #(TIME 6 8 7 9)))

;; Now consider an observation of make-noodles with a different agent. The constants
 ;; Joe and Sally are rigid designators, and therefore unequal.

Command: (EXPLAIN-OBSERVATION '(OBS-NOODLES4 MAKE-NOODLES
 (AGENT SALLY) #(TIME 6 8 7 9)))

T

:: Try to match this with the original make-sauce. It will fail, because agents differ.

Command: (MATCH-GRAPHS 'G2 'G4 'G2+G4)

... trying to match END249 with END267
... creating node END268
... checking constraints on END268
... END268 is okay
... checking constraints on END268
... END268 is okay
... trying to match PREPARE-MEAL248 with PREPARE-MEAL266
... creating node PREPARE-MEAL269
... equality constraint violated JOE = SALLY
... PREPARE-MEAL269 fails
... END268 fails
NIL

:: Lets check out the temporal constraints, now. We'll observe a boiling event
:: with time BEFORE the make-noodles event. This conflict will prevent a match.

Command: (EXPLAIN-OBSERVATION '(OBS-BOILING5 BOIL
(TIME #(TIME 1 1 2 2))) 'G5)

T

Command: (DRAW-GRAPH T)

graph G5

END273 NIL

?= PREPARE-MEAL272 ((TIME #(TIME -INF 1 2 +INF)))
?= MAKE-PASTA-DISH271 ((TIME #(TIME -INF 1 2 +INF)))
?= MAKE-PASTA-DISH270 ((TIME #(TIME -INF 1 2 +INF)))
S3 -> OBS-BOILING5 ((TIME #(TIME 1 1 2 2)))

:: Try to match this with G2 and G3. This will fail because of temporal constraint violation.

Command: (MATCH-GRAPHS 'G2+G3 'G5 'G2+G3+G5)

... trying to match END258 with END273
... creating node END274
... checking constraints on END274
... END274 is okay
... checking constraints on END274
... END274 is okay
... trying to match PREPARE-MEAL259 with PREPARE-MEAL272
... creating node PREPARE-MEAL275
... checking constraints on PREPARE-MEAL275
... PREPARE-MEAL275 is okay
... checking constraints on PREPARE-MEAL275
... PREPARE-MEAL275 is okay
... trying to match MAKE-PASTA-DISH260 with MAKE-PASTA-DISH271
... creating node MAKE-PASTA-DISH276
... checking constraints on MAKE-PASTA-DISH276
... MAKE-PASTA-DISH276 is okay
... checking constraints on MAKE-PASTA-DISH276
... MAKE-PASTA-DISH276 is okay
... trying to match MAKE-PASTA-DISH261 with MAKE-PASTA-DISH270
... creating node MAKE-PASTA-DISH277
... checking constraints on MAKE-PASTA-DISH277
... MAKE-PASTA-DISH277 is okay
... creating node MAKE-NOODLES278
... checking constraints on MAKE-NOODLES278
... MAKE-NOODLES278 is okay
... checking constraints on MAKE-NOODLES278
... MAKE-NOODLES278 is okay
... copying MAKE-NOODLES262 yields MAKE-NOODLES278
... creating node MAKE-SAUCE279
... checking constraints on MAKE-SAUCE279
... MAKE-SAUCE279 is okay
... checking constraints on MAKE-SAUCE279

... MAKE-SAUCE279 is okay
 ... copying MAKE-SAUCE263 yields MAKE-SAUCE279
 ... creating node BOIL280
 ... checking constraints on BOIL280
 ... BOIL280 is okay
 ... checking constraints on BOIL280
 ... BOIL280 is okay
 ... copying OBS-BOILING5 yields BOIL280
 ... checking constraints on MAKE-PASTA-DISH277
 ... time constraint violated #(TIME 6 8 7 9) BEFOREMEET #(TIME 1 1 2 2)
 ... MAKE-PASTA-DISH277 fails
 ... MAKE-PASTA-DISH276 fails
 ... PREPARE-MEAL275 fails
 ... END274 fails
 NIL

:: Now let's find a LATER boiling event. It will match okay.

Command: (EXPLAIN-OBSERVATION '(OBS-BOILING6 BOIL
 (TIME #(TIME 9 10 11 12))) 'G6)

T

Command: (MATCH-GRAPHS 'G2+G3 'G6 'G2+G3+G6)
 END285

Command: (DRAW-GRAPH T)
 graph G2+G3+G6
 END285 NIL

?= PREPARE-MEAL286 ((AGENT JOE) (TIME #(TIME -INF 5 11 +INF)))
 ?= MAKE-PASTA-DISH287
 ((AGENT JOE) (TIME #(TIME -INF 5 11 +INF)))
 ?= MAKE-PASTA-DISH288

```

((AGENT JOE) (TIME #(TIME -INF 5 11 +INF))
S3 -> BOIL291 ((TIME #(TIME 9 10 11 12)))
S2 -> MAKE-SAUCE290
((AGENT JOE) (TIME #(TIME 4 5 6 7)))
S1 -> MAKE-NOODLES289
((AGENT JOE) (TIME #(TIME 6 8 7 9)))

```

Operating System Examples

:: User enters: % copy foo bar.

Command: (EXPLAIN-OBSERVATION '(OBS-COPY1 COPY
(OLD FOO) (NEW BAR)) 'C1)

T

:: Could be part of rename by copy, or of modify file.

Command: (DRAW-GRAPH T)

graph C1

END294 NIL

?= RENAME297 ((NEW BAR) (OLD FOO))

?= RENAME-BY-COPY296 ((NEW BAR) (OLD FOO))

?= RENAME-BY-COPY295 ((NEW BAR) (OLD FOO))

COPY-ORIG-STEP -> OBS-COPY1 ((NEW BAR) (OLD FOO))

?= MODIFY293 ((FILE FOO))

?= MODIFY292 ((FILE FOO))

BACKUP-STEP -> OBS-COPY1 ((NEW BAR) (OLD FOO)) ^

:: User enters: % copy jack sprat.

Command: (EXPLAIN-OBSERVATION '(OBS-COPY2 COPY
(OLD JACK) (NEW SPRAT)) 'C2)

T

:: Try (and fail) to unify these commands. File names are rigid designators,
:: and so cannot be matched unless identical.

Command: (MATCH-GRAPHS 'C1 'C2 'C1+C2)

... trying to match END294 with END300
... creating node END304
... checking constraints on END304
... END304 is okay
... checking constraints on END304
... END304 is okay
... trying to match RENAME297 with RENAME303
... creating node RENAME305
... equality constraint violated BAR = SPRAT
... RENAME305 fails
... trying to match RENAME297 with MODIFY299
... trying to match MODIFY293 with RENAME303
... trying to match MODIFY293 with MODIFY299
... creating node MODIFY306
... equality constraint violated FOO = JACK
... MODIFY306 fails
... END304 fails
NIL

:: So, there must be two different plans going on.
:: User enters: % delete foo

Command: (EXPLAIN-OBSERVATION '(OBS-DELETE3 DELETE
(FILE FOO)) 'C3)

T

Command: (DRAW-GRAPH T)

graph C3

END310 NIL

 ?= MODIFY312 NIL

 ?= MODIFY311 NIL

 DELETE-BACKUP-STEP -> OBS-DELETE3 ((FILE FOO))

 ?= RENAME309 ((OLD FOO))

 ?= RENAME-BY-COPY308 ((OLD FOO))

 ?= RENAME-BY-COPY307 ((OLD FOO))

 DELETE-ORIG-STEP -> OBS-DELETE3 ((FILE FOO)) ^

:: This delete can unify with command 1, but not command 2

Command: (MATCH-GRAPHS 'C1 'C3 'C1+C3)

END313

Command: (DRAW-GRAPH T)

graph C1+C3

END313 NIL

 ?= RENAME314 ((NEW BAR) (OLD FOO))

 ?= RENAME-BY-COPY315 ((NEW BAR) (OLD FOO))

 ?= RENAME-BY-COPY316 ((NEW BAR) (OLD FOO))

 DELETE-ORIG-STEP -> DELETE318 ((FILE FOO))

 COPY-ORIG-STEP -> COPY317 ((NEW BAR) (OLD FOO))

:: Command 2 and 3 cannot be part of the end plan

Command: (MATCH-GRAPHS 'C2 'C3 'C2+C3)

... trying to match END300 with END310

... creating node END321

... checking constraints on END321

... END321 is okay

... checking constraints on END321

... END321 is okay
... trying to match RENAME303 with MODIFY312
... trying to match RENAME303 with RENAME309
... creating node RENAME322
... equality constraint violated JACK = FOO
... RENAME322 fails
... trying to match MODIFY299 with MODIFY312
... creating node MODIFY323
... checking constraints on MODIFY323
... MODIFY323 is okay
... checking constraints on MODIFY323
... MODIFY323 is okay
... trying to match MODIFY298 with MODIFY311
... creating node MODIFY324
... checking constraints on MODIFY324
... MODIFY324 is okay
... creating node COPY325
... checking constraints on COPY325
... COPY325 is okay
... checking constraints on COPY325
... COPY325 is okay
... copying OBS-COPY2 yields COPY325
... creating node DELETE326
... checking constraints on DELETE326
... DELETE326 is okay
... checking constraints on DELETE326
... DELETE326 is okay
... copying OBS-DELETE3 yields DELETE326
... checking constraints on MODIFY324
... equality constraint violated FOO = SPRAT
... MODIFY324 fails
... MODIFY324 fails
... MODIFY323 fails
... trying to match MODIFY299 with RENAME309

... END321 fails
NIL

Language Examples

;; Joe says to Sally: "Can you give me the salt?"

Command: (EXPLAIN-OBSERVATION '(TALK1 SURFACE-QUESTION
 (SPEAKER JOE)
 (HEARER SALLY)
 (TIME #(TIME 4 4 5 5))
 (CONTENT (CAN SALLY (GAVE SALLY JOE SALT))))
 UTTER1)

... creating node TALK1
... searching up from TALK1
... creating node DIRECT-REQUEST360
... considering TALK1 as S of DIRECT-REQUEST360
... checking constraints on DIRECT-REQUEST360
... DIRECT-REQUEST360 is okay
... searching up from DIRECT-REQUEST360
... creating node DIRECT-REQUEST361
... searching up from DIRECT-REQUEST361
... creating node REQUEST362
... searching up from REQUEST362
... creating node OBTAIN-BY-ASKING363
... considering REQUEST362 as S1 of OBTAIN-BY-ASKING363
... checking constraints on OBTAIN-BY-ASKING363
... equality constraint violated GAVE = INFORMEDIF
... OBTAIN-BY-ASKING363 fails
... OBTAIN-BY-ASKING363 fails
... creating node FINDOUT-BY-ASKING364
... considering REQUEST362 as S1 of FINDOUT-BY-ASKING364

... checking constraints on FINDOUT-BY-ASKING364
... FINDOUT-BY-ASKING364 is okay
... searching up from FINDOUT-BY-ASKING364
... creating node FINDOUT-BY-ASKING365
... searching up from FINDOUT-BY-ASKING365
... creating node FINDOUT366
... searching up from FINDOUT366
... creating node END367
... searching up from END367
... creating node INDIRECT-REQUEST368
... considering TALK1 as S of INDIRECT-REQUEST368
... checking constraints on INDIRECT-REQUEST368
... INDIRECT-REQUEST368 is okay
... searching up from INDIRECT-REQUEST368
... creating node INDIRECT-REQUEST369
... searching up from INDIRECT-REQUEST369
... creating node REQUEST370
... searching up from REQUEST370
... creating node OBTAIN-BY-ASKING371
... considering REQUEST370 as S1 of OBTAIN-BY-ASKING371
... checking constraints on OBTAIN-BY-ASKING371
... OBTAIN-BY-ASKING371 is okay
... searching up from OBTAIN-BY-ASKING371
... creating node OBTAIN-BY-ASKING372
... searching up from OBTAIN-BY-ASKING372
... creating node OBTAIN373
... searching up from OBTAIN373
... merging graph at END367
... creating node FINDOUT-BY-ASKING374
... considering REQUEST370 as S1 of FINDOUT-BY-ASKING374
... checking constraints on FINDOUT-BY-ASKING374
... equality constraint violated INFORMEDIF = GAVE
... FINDOUT-BY-ASKING374 fails
... FINDOUT-BY-ASKING374 fails

T

:: The statement is ambiguous, as we see:

Command: (DRAW-GRAPH T)

graph UTTER1

END367 NIL

?= OBTAIN373

((OBJECT SALT)

(AGENT JOE)

(TIME #(TIME -INF 4 5 +INF)))

(PRETIME #(TIME -INF 4 -INF +INF))

?= OBTAIN-BY-ASKING372

((AGENT JOE) (OBJECT SALT)

(TIME #(TIME -INF 4 -INF +INF))

(PRETIME #(TIME -INF 4 -INF +INF)))

?= OBTAIN-BY-ASKING371

((AGENT JOE) (OBJECT SALT)

(TIME #(TIME -INF 4 -INF +INF))

(PRETIME #(TIME -INF 4 -INF +INF)))

S1 -> REQUEST370

((REQGOAL (GAVE SALLY JOE SALT))

(HEARER SALLY)

(SPEAKER JOE)

(TIME #(TIME 4 4 5 5)))

?= INDIRECT-REQUEST369

((SPEAKER JOE) (HEARER SALLY)

(REQGOAL (GAVE SALLY JOE SALT))

(TIME #(TIME 4 4 5 5)))

?= INDIRECT-REQUEST368

((SPEAKER JOE)

(HEARER SALLY)

(REQGOAL (GAVE SALLY JOE SALT))

(TIME #(TIME 4 4 5 5)))

S -> TALK1

((CONTENT (CAN SALLY
(GAVE SALLY JOE SALT)))
(HEARER SALLY)
(SPEAKER JOE)
(TIME #(TIME 4 4 5 5)))

?= FINDOUT366

((INFO (CAN SALLY (GAVE SALLY JOE SALT)))
(AGENT JOE)
(PRETIME #(TIME -INF 4 -INF +INF))
(TIME #(TIME -INF 4 5 +INF)))

?= FINDOUT-BY-ASKING365

((AGENT JOE)
(INFO (CAN SALLY (GAVE SALLY JOE SALT)))
(TIME #(TIME -INF 4 5 +INF))
(PRETIME #(TIME -INF 4 -INF +INF)))

?= FINDOUT-BY-ASKING364

((AGENT JOE)
(INFO (CAN SALLY (GAVE SALLY JOE SALT)))
(TIME #(TIME -INF 4 5 +INF))
(PRETIME #(TIME -INF 4 -INF +INF)))

S1 -> REQUEST362

((REQGOAL (INFORMEDIF SALLY JOE
(CAN SALL (GAVE SALLY JOE SALT)))
(HEARER SALLY)
(SPEAKER JOE)
(TIME #(TIME 4 4 5 5)))

?= DIRECT-REQUEST361

((SPEAKER JOE)
(HEARER SALLY)
(REQGOAL (INFORMEDIF SALLY JOE
(CAN SALLY (GAVE SALLY JOE SALT)))
(TIME #(TIME 4 4 5 5)))

?= DIRECT-REQUEST360

```

((SPEAKER JOE)
(HEARER SALLY)
(REQGOAL (INFORMEDIF SALLY JOE
          (CAN SALLY (GAVE SALLY JOE SALT))))
(TIME #(TIME 4 4 5 5)))
S -> TALK1
((CONTENT (CAN SALLY
          (GAVE SALLY JOE SALT)))
(HEARER SALLY)
(SPEAKER JOE)
(TIME #(TIME 4 4 5 5)))

```

;; There are 2 (of 4) alternatives not eliminated: an attempt to find out if Sally
 ;; can give Joe the salt, or an attempt to obtain the salt.

;; Now lets add the fact that at all times, Joe knows if Sally can give him the salt.

The implementation did not include an explicit Holds predicate; instead, it included a fuzzy time constraint as the second argument to fact predicates. The following assertion means

$$\forall t \in (-\infty \ -\infty \ +\infty \ +\infty) .$$

Holds(t, (knowif Joe (can Sally (gave Sally Joe Salt))))

Command: (ADD-FACT '(KNOWIF #(TIME :-INF :-INF :+INF :+INF) JOE
 (CAN SALLY (GAVE SALLY JOE SALT))))

T

;; Repeat the example. Joe says to Sally: "Can you give me the salt?"

Command: (EXPLAIN-OBSERVATION '(TALK2 SURFACE-QUESTION
 (SPEAKER JOE) (HEARER SALLY)
 (CONTENT (CAN SALLY (GAVE SALLY JOE SALT))))

'UTTER2)

... creating node TALK2

... searching up from TALK2
 ... creating node DIRECT-REQUEST375
 ... considering TALK2 as S of DIRECT-REQUEST375
 ... checking constraints on DIRECT-REQUEST375
 ... DIRECT-REQUEST375 is okay
 ... searching up from DIRECT-REQUEST375
 ... creating node DIRECT-REQUEST376
 ... searching up from DIRECT-REQUEST376
 ... creating node REQUEST377
 ... searching up from REQUEST377
 ... creating node OBTAIN-BY-ASKING378
 ... considering REQUEST377 as S1 of OBTAIN-BY-ASKING378
 ... checking constraints on OBTAIN-BY-ASKING378
 ... equality constraint violated GAVE = INFORMEDIF
 ... OBTAIN-BY-ASKING378 fails
 ... OBTAIN-BY-ASKING378 fails
 ... creating node FINDOUT-BY-ASKING379
 ... considering REQUEST377 as S1 of FINDOUT-BY-ASKING379
 ... checking constraints on FINDOUT-BY-ASKING379
 ... fact constraint violated (NEVER #(TIME -INF 4 -INF +INF) KNOWIF JOE
 (CAN SALLY (GAVE SALLY JOE SALT)))
 ... FINDOUT-BY-ASKING379 fails
 ... REQUEST377 fails
 ... DIRECT-REQUEST376 fails
 ... DIRECT-REQUEST375 fails
 ... creating node INDIRECT-REQUEST380
 ... considering TALK2 as S of INDIRECT-REQUEST380
 ... checking constraints on INDIRECT-REQUEST380
 ... INDIRECT-REQUEST380 is okay
 ... searching up from INDIRECT-REQUEST380
 ... creating node INDIRECT-REQUEST381
 ... searching up from INDIRECT-REQUEST381
 ... creating node REQUEST382
 ... searching up from REQUEST382

... creating node OBTAIN-BY-ASKING383
 ... considering REQUEST382 as S1 of OBTAIN-BY-ASKING383
 ... checking constraints on OBTAIN-BY-ASKING383
 ... OBTAIN-BY-ASKING383 is okay
 ... searching up from OBTAIN-BY-ASKING383
 ... creating node OBTAIN-BY-ASKING384
 ... searching up from OBTAIN-BY-ASKING384
 ... creating node OBTAIN385
 ... searching up from OBTAIN385
 ... creating node END386
 ... searching up from END386
 ... creating node FINDOUT-BY-ASKING387
 ... considering REQUEST382 as S1 of FINDOUT-BY-ASKING387
 ... checking constraints on FINDOUT-BY-ASKING387
 ... equality constraint violated INFORMEDIF = GAVE
 ... FINDOUT-BY-ASKING387 fails
 ... FINDOUT-BY-ASKING387 fails
 T

:: The only interpretation is the indirect request.

Command: (DRAW-GRAPH T)

graph UTTER2

?= OBTAIN385

((OBJECT SALT)

(AGENT JOE)

(TIME #(TIME -INF 4 5 +INF)))

(PRETIME #(TIME -INF 4 -INF +INF))

?= OBTAIN-BY-ASKING384

((AGENT JOE) (OBJECT SALT)

(TIME #(TIME -INF 4 -INF +INF))

(PRETIME #(TIME -INF 4 -INF +INF)))

?= OBTAIN-BY-ASKING383

((AGENT JOE) (OBJECT SALT)

(TIME #(TIME -INF 4 -INF +INF))
(PRETIME #(TIME -INF 4 -INF +INF)))

S1 -> REQUEST382

((REQGOAL (GAVE SALLY JOE SALT))
(HEARER SALLY)
(SPEAKER JOE)
(TIME #(TIME 4 4 5 5)))

?= INDIRECT-REQUEST381

((SPEAKER JOE) (HEARER SALLY)
(REQGOAL (GAVE SALLY JOE SALT))
(TIME #(TIME 4 4 5 5)))

?= INDIRECT-REQUEST380

((SPEAKER JOE)
(HEARER SALLY)
(REQGOAL (GAVE SALLY JOE SALT))
(TIME #(TIME 4 4 5 5)))

S -> TALK2

((CONTENT (CAN SALLY
(GAVE SALLY JOE SALT)))
(HEARER SALLY)
(SPEAKER JOE)
(TIME #(TIME 4 4 5 5)))

Appendix F

Details of Correctness

Proof of Explain

The key to the correctness of **explain** – the fact that *all* possible interpretations are considered – lies in the interaction of the **redundant** subroutine and the definition of the set **Uses**. **Redundant** blocks consideration of uses of an event which are covered by other, more apt uses. For example, if the primary event to be explained is of type **MakeMarinara**, the use (**MakeSauce**, **s2**, **MakePastaDish**) is redundant, while if the primary event to be explained is of type **MakeSauce**, the use (**MakeMarinara**, **s2**, **MakeSpaghettiMarinara**) is redundant.

Following is the statement which must be shown to hold in order for **explain** to be correct.

Statement of the Problem

Consider the invocation

explain(E_0 , D , $\emptyset$, **true**, E_0)

For every (event type) E_u such that E_u has an r -direct component of type E_c which is compatible with E_0 , and for every basic specialization* E_b of E_u whose instances could possibly have an r -component of E_0 :

the algorithm creates a node $E_a(n_a)$ such that E_a abstracts* E_b , whose r -component $E_a(n_a)$ is a node of type compatible with E_0 .

Multiple Inheritance Proof

By inspection we see that **explain** does create a node $E_c(n_c)$, which holds one of the following relations to $E_0(n_0)$.

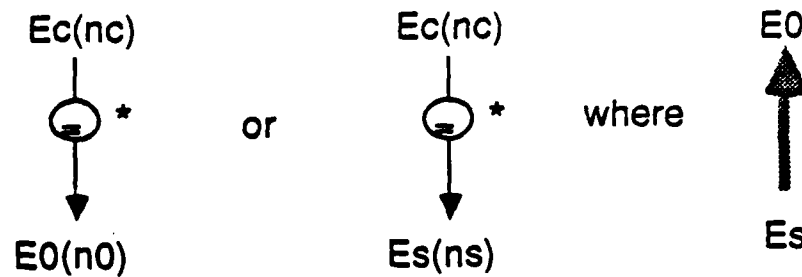


figure F.1: Relation of $E_c(nc)$ to $E_0(n_0)$

If the component/use inference step performs a recursive call for use (E_c, r, E_u) , then the condition is satisfied by $E_u(n_u)$ itself. If this doesn't occur, then the **redundant** test must have blocked the use. So consider the ways in which **redundant** could be satisfied.

Case 1. (E_c, r, E_u) abstracts a use for a type which abstracts* **primary**. There must be a least abstract such type, call it E_1 , with use (E_1, r, E_d) . This is illustrated below.

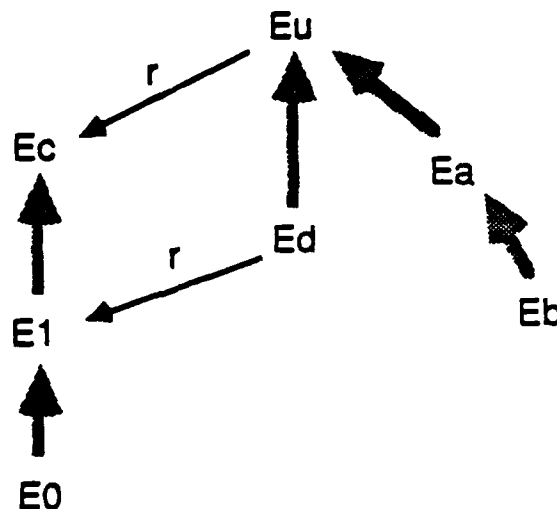


figure F.2: Case 1

When **explain** visited $E_1(n_1)$ it must have considered all **r**-uses of E_1 , since by inspection **redundant** could not block them. Now we claim that there are **r**-uses which fulfil the condition.

Let E_b be the specialization of E_u in question. Consider the generation of Uses. Let E_a be the most general abstraction* of E_b such that:

- i. All specializations* of E_a could have an r -th component of type E_1
- ii. $(E_1, r, E_a) \in U_\alpha$

By definition of E_a , (E_1, r, E_a) appears in Uses. Thus node $E_a(n_a)$, with roleval (r, n_1) , is exactly the node needed for the proof.

Case 2. (E_c, r, E_u) specializes a use for a type which specializes* primary. There must be a most general such type, call it E_1 , with use (E_1, r, E_a) . This is illustrated below.

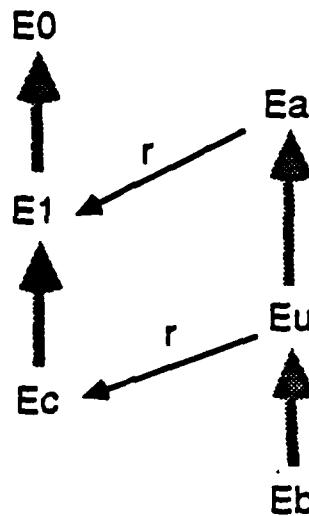


figure F.3: Case 2

By inspection we see that the redundant test cannot block this use, so node $E_a(n_a)$ is the desired node, because E_u abstracts* E_b and E_a abstracts* E_u .

Case 3. (E_c, r, E_u) specializes a use for a type which abstracts E_0 and E_c does not abstract* or specialize* E_0 . There must be a least general such type, call it E_1 , with use (E_1, r, E_d) . Let E_2 be the least general abstraction* of E_0 which has a use (E_2, r, E_f) which specializes* (E_1, r, E_d) . (Possibly $E_2 = E_1$.) This is illustrated below.

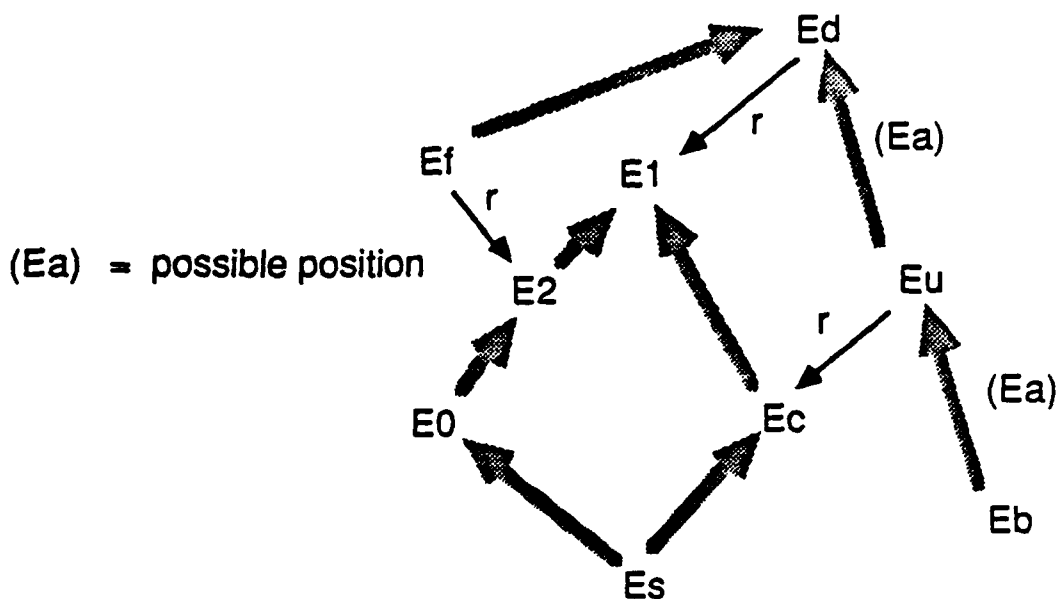


figure F.4: Case 3

Let E_a be the most general abstraction* of E_b such that

- i. All specializations* of E_a could possibly have an r -th component of type E_2 .
- ii. $(E_2, r, E_a) \in U_\alpha$

ii. $(E_2, r, E_a) \in U_\alpha$

By definition of E_a , (E_2, r, E_a) appears in *Uses*. By inspection we see that this use cannot be blocked by *redundant*, so $E_a(n_a)$, with *roleval* (r, n_2) , is the desired node.

Single Inheritance Proof

Consider the single-inheritance version of `redundant`. There are two conditions under which `redundant` returns true:

Case 1. (etype, r, utype) abstracts a use for some member of visited. Because abstractions of primary are considered before specializations of primary, it must be the case that (etype, r, utype) abstracts a use for a type which abstracts* primary: that is, the first case above.

Case 2. (etype, r, utype) specializes a use for some member of visited. Because visited is grown depth first, it must be the case that (etype, r, utype) specializes a use for a type which specializes* primary: that is, the second case above.

This completes the details of the proof of correctness of explain.

A Model for Concurrent Actions having Temporal Extent

Richard N. Pelavin
Philips Laboratories
North American Philips Corporation
Briarcliff Manor, N.Y. 10510

James F. Allen
Computer Science Department
University of Rochester
Rochester, N.Y. 14607

Abstract

In this paper we present a semantic model that is used to interpret a logic that represents concurrent actions having temporal extent. In an earlier paper [Pelavin and Allen, 1986] we described how this logic is used to formulate planning problems that involve concurrent actions and external events. In this paper we focus on the semantic structure. This structure provides a basis for describing the interaction between actions, both concurrent and sequential, and for composing simple actions to form complex ones. This model can also treat actions that are influenced by properties that hold and events that occur during the time that the action is to be executed. Each model includes a set of world-histories, which are complete worlds over time, and a function that relates world-histories that differ solely on the account of an action executed at a particular time. This treatment derives from the semantic theories of conditionals developed by Stalnaker [Stalnaker, 1968] and Lewis [Lewis, 1973].

I. Introduction

One of the most successful approaches to representing events and their effects in Artificial Intelligence has been situation calculus [McCarthy and Hayes, 1969]. In this logic, an event is modeled by a function from situation, i.e. instantaneous snapshot of the world, to situation. This function captures the state changes produced by the event in different situations.

A deficiency of this representation is that simultaneous events cannot be directly modeled; one cannot describe the result produced by two events initiated in the same situation (see, however, Georgeff [Georgeff, 1986] who extends and modifies situation calculus so this can be done). Another deficiency is that situation calculus does not capture what is happening while an event is occurring. Thus, one cannot directly treat events that are affected by conditions that hold during execution, such as the event "sailing across the lake"

which can occur only if the wind is blowing while the sailing is taking place.

Allen [Allen, 1984] and McDermott [McDermott, 1982] have put forth logics that represent simultaneous events and events with temporal extent. Allen develops a linear time model based on intervals, i.e. contiguous chunks of time. McDermott describes a branching time model where a set of instantaneous states are arranged into a tree that branches into the future. McDermott uses the term "interval" to refer to a convex set of states that lie along a branch in the tree of states.

In both logics, an event is equated with the set of intervals over which the event occurs. Properties, which capture static aspects of the world, are treated in a similar manner. Each property is equated with the set of intervals over which the property holds. Simultaneous events can be described by stating that two events occur over intervals that overlap in time. One can also describe the properties that hold and the events that occur while some event takes place.

Although these logics overcome some of the deficiencies of situation calculus, they are not adequate for reasoning about actions and forming plans. These logics lack a structure analogous to the result function in situation calculus that describes the result of executing different actions in different contexts. In situation calculus, the context is given by the situation in which an action is to be initiated. At each situation, one can describe whether an action can be successfully executed and describe whether an action negates some property or does not affect it. This structure also provides a simple basis for constructing complex actions, i.e. sequences of actions.

Without extension, similar statements cannot be made in Allen's and McDermott's logics. For example, one cannot describe that an action does not affect some property or event, such as stating that raising one's arm does not affect whether it is raining out. One cannot represent that some action can be executed only under certain conditions, such as stating that the agent can edit a document during interval *i* only if the text editor is operational during *i*.

Allen's and McDermott's logics do not provide a basis for relating the conditions under which a set of actions, concurrent or sequential, can be executed together to the conditions under which the actions, making up the set, can be executed individually. Whether two actions can be executed together depends on how they interact. For example, one may be able to execute two actions individually, but not concurrently, such as "moving one's hand up" and

This paper describes work done in the Computer Science Department at the University of Rochester. It was supported in part by the National Science Foundation under grant DCR-8502481, the Air Force Systems Command, Rome Air Development Center, and the Air Force Office of Scientific Research under contract number F30602-85-C-0008. This contract supports the North East Artificial Intelligence Consortium (NAIC).

"moving one's hand down". It might be the case that two actions can be executed together only under certain conditions, such as two concurrent actions that share the same type of resource. Allen's and McDermott's logics can express "if actions a_1 and a_2 both occur during i , then there must be at least two resources available during i ". These logics, however, cannot distinguish whether "there are at least two resources available during i " is a necessary condition that must hold in order to execute a_1 and a_2 together, or whether this condition is an effect produced by the joint execution of a_1 and a_2 . A detailed discussion of these issues is given in [Pelavin, 1987].¹

To remedy these problems, we develop a semantic model that contains a structure analogous to the result function in situation calculus. In our models, *world-histories* and *action instances* take the place of situations and actions. A world-history refers to a complete world over time, rather than an instantaneous snapshot. An action instance, refers to an action to be performed at a specified time. A world-history serves as the context in which the execution of an action instance is specified. This enables us to model the influence of conditions that may hold during the time that an action instance is to be executed, and, as we will see, provides a simple basis for modeling concurrent interactions and for defining the joint execution of a set of action instances.

To describe these models, we extend Allen's language, which is a first order language, with two modal operators. In this paper, we only describe the underlying semantic structure and do not discuss the syntax or interpretation of this modal language. Moreover, we focus on the portion of the model that pertains to modeling actions, after briefly describing the other components in the model structure. The reader interested in the language, axiomatics, or other details omitted in this paper can refer to [Pelavin, 1987] and [Pelavin and Allen, 1988].

II. Overview of the model structure

In each model, a set of world-histories and set of temporal intervals are identified. Each temporal interval picks out a common time across the set of world-histories. The intervals are arranged by the MEETS relation to form a global date line. The relation $MEETS(i_1, i_2)$ is true if interval i_1 is immediately prior to interval i_2 . In [Allen and Hayes, 1985], it is shown that all temporal interval relations, such as "overlaps to the right" and "starts", can be defined in terms of MEETS.

The model identifies the set of properties and events that hold (occur) at various times in the different world-histories. Formally, events and properties are sets of ordered pairs, each formed by an interval and a world-history. If $\langle i, h \rangle \in ev$, then event ev occurs during interval i in world-history h . Similarly, if $\langle i, h \rangle \in pr$, then property pr holds during interval i in world-history h . To capture the relation "if property pr holds over an interval i then pr holds over all intervals contained in i ", we restrict the models so

¹ For example, in [Pelavin, 1987] we show why a branching time model cannot be used to interpret "action a_1 can be executed during time i " if we want to treat actions, such as "sailing", that are influenced by conditions that hold during execution.

that if i_1 is contained in i_2 and $\langle i_2, h \rangle \in pr$, then $\langle i_1, h \rangle \in pr$.

World-histories are arranged into trees that branch into the future by the R accessibility relation which takes an interval and two world-histories as arguments. Intuitively, $R(i, h_1, h_2)$ means that h_1 and h_2 share a common past through the end of interval i and are possible with respect to each other at i . This structure is identical to one found in [Haas, 1985] with the exception that Haas uses a time point to relate world-theories, rather than the end of an interval. Constraints are placed on R to insure that i) it is an equivalence relation for a fixed interval, ii) if $R(i, h_1, h_2)$, then h_1 and h_2 agree on all events and properties that end before or at the same time as i , and iii) if $R(i, h_1, h_2)$, then $R(i_2, h_1, h_2)$ for all intervals i_2 that end at the same time as or before i .

In situation calculus, the execution of an action is given with respect to a situation, and an action is modeled as a function from situation to situation. In our model, the execution of an action instance is given with respect to a world-history, and an action instance is modeled as a function from world-history to set of world-histories. The rest of the paper is devoted to describing this function and showing how a function associated with a set of action instances can be constructed from the functions associated with its members.

In this paper, we will only discuss a type of action instance called a *basic action instance*. Basic actions [Goldman, 1970] refer to actions that are primitive in the sense that all non-basic actions are brought about by performing one or more basic actions under appropriate conditions. In [Pelavin, 1987], we describe how all other action instances (which we refer to as "plan instances") are defined in terms of basic action instances.

III. The F_{cl} function

The result of executing a basic action instance with respect to a world-history is given by the F_{cl} function. F_{cl} takes a basic action instance bai and a world-history h as arguments and yields a nonempty set of world-histories that "differ from h solely on the account of the occurrence of bai ". Equivalently, we say that the world-histories belonging to $F_{cl}(bai, h)$ are the "closest world-histories" to h where basic action instance bai occurs. The term "closest" is a vestige from Stalnaker's [Stalnaker, 1968] and Lewis' [Lewis, 1973] semantic theories of conditionals from which our treatment derives.

In the remainder of this section, we explain what we mean by "differing solely on the account of the occurrence of a basic action instance" and present the constraints that are imposed on F_{cl} in accordance with these intuitions. Very briefly, if h_2 belongs to $F_{cl}(bai, h)$, then h and h_2 will coincide on all conditions that are not affected by the occurrence of bai . This includes conditions out of the agent's control, such as whether or not it is raining during some interval, and conditions that only refer to times that end before bai . One reason that F_{cl} yields a set of world-histories, rather than a single one, is to provide for non-deterministic basic actions. Another reason for treating F_{cl} as a set is explained later.

We use a term of the form " $ba@i$ " to refer to a basic action instance whose time of occurrence is i . The treatment of $F_{cl}(ba@i, h)$ is trivial when $ba@i$ occurs in h . In this case, $F_{cl}(ba@i, h)$ is equal to $\{h\}$ reflecting the principle that a world-history is closer to itself than any other world-history. This is captured by the following constraint which is imposed on our models:

BA1)

For all basic action instances ($ba@i$),
and world-histories (h), if $h \in OC(ba@i)$,
then $F_{cl}(ba@i, h) = \{h\}$

where $OC(ba@i)$ is the set of world-histories
in which $ba@i$ occurs

$F_{cl}(ba@i, h)$ is also set to $\{h\}$ when $ba@i$'s standard conditions do not hold in h . The term "standard conditions" is taken from Goldman [Goldman, 1970] although we use it in more general way. A basic action's standard conditions are conditions that must hold in order to execute the action. For example, the standard conditions for "the agent moves its right arm up during time i " include the condition that the arm is not broken during time i . We also use standard conditions to refer to the conditions under which a move is legal when modeling a board game.

If $ba@i$'s standard conditions do not hold in h , then $F_{cl}(ba@i, h)$, which equals $\{h\}$, contains a world-history in which $ba@i$ does not occur. In effect, if $ba@i$'s standard conditions do not occur in h , we are not defining "the closest world-history to h where $ba@i$ occurs". We treat the lack of standard conditions this way because we want to restrict F_{cl} so that if $h2$ belongs to $F_{cl}(ba@i, h)$ then $h2$ and h agree on all conditions that are not affected, directly or indirectly, by $ba@i$. This restriction would be violated if $ba@i$'s standard conditions did not hold in h , but $F_{cl}(ba@i, h)$ contained a world-history $h2$ where $ba@i$ occurs. This stems from an assumption that a basic action cannot affect whether or not its own standard conditions hold.

$F_{cl}(ba@i, h)$ yields a non-trivial result when $ba@i$'s standard conditions hold in h , but $ba@i$ does not occur in h . In this case, all the members belonging to $F_{cl}(ba@i, h)$ differ from h and $ba@i$ occurs in all these world-histories. Consequently, we impose the following constraint:

BA2)

For all world-histories (h and $h2$)
and basic action instances ($ba@i$),
if $F_{cl}(ba@i, h) \neq \{h\}$ then
 $F_{cl}(ba@i, h) \subseteq OC(ba@i)$

Typically, when $h2$ belongs to $F_{cl}(ba@i, h)$ and $h2$ is distinct from h (which we will assume in the rest of this section), the two world-histories will differ on more than the status of " $ba@i$ occurs". We assume that the set of world-histories adhere to a set of laws that govern the relations between events, properties, and other objects in the world-histories. A world-history formed by just modifying h to make " $ba@i$ occurs" true may violate some laws. Consequently, h and $h2$ will also differ on some conditions that are related, directly or indirectly, to " $ba@i$ occurs" by some set of laws.

As an example, suppose that property $pr2$ does not hold during interval $i2$ in h , but there is a law that entails that if $ba@i$ occurs then $pr2$ holds during $i2$. Consequently, h and $h2$ must differ on the status of " $pr2$ holds during $i2$ " since $ba@i$ occurs in $h2$. World-histories h and $h2$ may also differ on conditions that are indirectly affected by $ba@i$. Suppose that there is a second law that entails that if $pr2$ holds during $i2$ then $pr3$ holds during $i3$. If $pr3$ does not hold during $i3$ in h , then h and $h2$ will also differ on this condition.

As a second example, consider a law that entails that $ba@i$ and $ba2@i$ cannot occur together. Thus, if $ba2@i$ occurs in h , any world-history $h2$ belonging to $F_{cl}(ba@i, h)$ will differ from h because $ba2@i$ does not occur in $h2$. This type of relation, as we will see, forms the basis for detecting interference between basic action instances and is used when composing basic action instances together.

We assume that the difference between h and $h2$ are minimal in that changes are only made in going from h to $h2$ to satisfy laws that would be violated if these changes were not made. They agree on all other conditions. This includes conditions out of the agent's control such as whether or not it is raining out. We also constrain our models so that h and $h2$ agree on all conditions that refer to times that are prior to $ba@i$'s time of occurrence. This is captured by a constraint relating F_{cl} to the R relation which is given as follows:

BA-R1)

For all world-histories ($h1$ and $h2$),
basic action instances ($ba@i$), and intervals ($i0$),
if $h2 \in F_{cl}(ba@i, h1)$ and $MEETS(i0, i)$, then $R(i0, h, h2)$

BA-R1 entails the relation that two world-histories differing on the occurrence of $ba@i$ must coincide on all conditions that end before the beginning of interval i . This restriction presupposes that there are no laws specifying whether or not a basic action instance, whose standard conditions hold, occurs.

One reason why $F_{cl}(ba@i, h)$ yields a set of world-histories, instead of a single one, is that there may be many ways to minimally modify h to account for $ba@i$'s occurrence. For example, suppose that only two of the three basic action instances, $ba1@i$, $ba2@i$, and $ba3@i$, can be executed together. Also assume that both $ba2@i$ and $ba3@i$ occur in h . In this case, $F_{cl}(ba1@i, h)$ will contain (at least) two world-histories: one where both $ba1@i$ and $ba2@i$ occur, but not $ba3@i$, and another where $ba1@i$ and $ba3@i$ occur, but not $ba2@i$.

It is important to emphasize that the F_{cl} function is part of the semantic model and thus there is no need to precisely specify this function when reasoning in our logic. We describe the world using a set of sentences in our language (which is described in [Pelavin and Allen, 1986] and [Pelavin, 1987]). Typically, a set of sentences only partially describe a model; there may be many models that satisfy a set of sentences. The F_{cl} function provides a simple underlying structure to interpret sentences that describe what a basic action instance affects and does not affect with respect to a context that may include conditions that hold while the basic action instance is to be executed. As we will see, it also provides a simple basis for modeling basic action instance interactions and for treating the joint execution of a set of basic action instances.

IV. Composing action instances

The result of executing a set of basic action instances together is computed from the individual members in the set. In other words, F_{cl} applied to a set of basic action instances *bai-set* is defined in terms of F_{cl} applied individually to each member in *bai-set*. In this section, we will let F_{cl} take a set of basic actions instances as an argument, rather than a single one; F_{cl} applied to the singleton set {bai} is to be treated as we described F_{cl} applied to *bai* in the last section.

Any set of basic action instances can be composed together regardless of their temporal relation. Moreover, the definition of F_{cl} applied to *bai-set* does not need to be conditionalized on the temporal relations between the members of *bai-set*. So for example, the composition of two concurrent basic action instances is defined in the same way as the composition of two basic action instances that do not overlap in time.

The following notation is introduced to succinctly present the definition of F_{cl} applied to a basic action instance set and to present two related constraints.

The constructor function "*" combines two functions from H to 2^H to form a function from H to 2^H , where H denotes a set of world-histories:

$$fx * fy(h) =_{def} \bigcup_{h' \in f(h)} fy(h')$$

The set of composition functions of a basic action instance set is recursively defined by:

- i) A singleton basic action instance set {bai} has one composition function: $\lambda h.F_{cl}(\{bai\},h)$
- ii) The composition functions of a basic action instance set *bai-set* with more than one element: $\{bai * cmp \mid bai \in \text{bai-set and } cmp \text{ is a composition function of } (bai\text{-set} - bai)\}$

If *cmp* is a composition function of *bai-set*, then *cmp*(*h*) yields the set of world-histories that would be reached by successively modifying *h* by the basic action instances belonging to *bai-set* in some order.

ALL-OC relates world-histories and composition functions:

$$\text{For any composition function of } bai\text{-set } (cmp), \\ \text{ALL-OC}(h,cmp) =_{def} cmp(h) \subseteq OC(bai\text{-set})$$

If *cmp* is a composition function of *bai-set*, then *ALL-OC*(*h*,*cmp*) is true iff the result of modifying *h* successively by all the members in *bai-set*, in the order implicit in *cmp*, yields a set of world-histories where all the members in *bai-set* occur.

Finally, the definition of F_{cl} and constraints BA-CMP1 and BA-CMP2 are given by:

$$F_{cl}(bai\text{-set},h) =_{def} \begin{cases} cmp(h) & \text{If there exists a composition} \\ & \text{function of } bai\text{-set } (cmp) \\ & \text{such that } \text{ALL-OC}(h,cmp). \\ \{h\} & \text{Otherwise} \end{cases}$$

BA-CMP1)

For all world-histories (*h*) and basic action instance sets (*bai-set*), if there exists two composition functions of *bai-set* (*cmp1* and *cmp2*) such that *ALL-OC*(*h*,*cmp1*) and *ALL-OC*(*h*,*cmp2*), then *cmp1*(*h*) = *cmp2*(*h*)

BA-CMP2)

For all world-histories (*h*) and composition functions (*cmp1* and *cmp2*), if *ALL-OC*(*h*,*cmp2*) and *ALL-OC*(*h*,*cmp1***cmp2*) then *ALL-OC*(*h*,*cmp2***cmp1*)

In the following discussion, we examine BA-CMP1, BA-CMP2, and the definition of F_{cl} (*bai-set*) for the case where *bai-set* consists of two basic action instances, *bai1* and *bai2*, that yield unique closest world-histories when F_{cl} is applied to either of them at any world-history. In [Pelavin, 1987], a detailed explanation is provided for the other cases, such as when *bai-set* contains three or more members.

The set {*bai1*,*bai2*} has two composition functions, which we will denote by *bai1***bai2* and *bai2***bai1*. *bai1***bai2*(*h*) yields a singleton set containing the world-history obtained by modifying *h*, first by *bai1*, then by *bai2*. *bai2***bai1*(*h*) yields a singleton set containing the world-history obtained by modifying *h*, first by *bai2*, then by *bai1*. It is important to keep in mind that *bai1* and *bai2* have fixed times associated with them and consequently may have any temporal relation. Thus, *bai1***bai2*(*h*) does not necessarily describe the results of executing *bai1* before *bai2*, since *bai2* may be prior to or concurrent with *bai1*.

Let us first consider the case where *bai1*'s and *bai2*'s standard conditions hold at all world-histories. We say that *bai1* and *bai2* interfere at world-history *h* if they cannot be executed together in the context given by *h*. If they interfere, we set $F_{cl}(\{bai1,bai2\},h)$ to {*h*}, treating {*bai1*,*bai2*} as if its standard conditions do not hold at *h*. As an example, "move right hand up during *i*" and "move right hand down during *i*" are basic action instances that interfere at all world-histories (when modeling a typical world). Conversely, "move right hand up during *i*" and "move left hand down during *i*" do not interfere at any world-history.

We may also model basic action instances that conditionally interfere, ones that interfere at some world-histories but not at others. For example, if two concurrent basic action instances share the same type of resource, they interfere only at world-histories where there is not enough of this resource available during their time of execution. It is important to note that interference is defined relative to world-histories. Consequently, whether two or more basic actions interfere can depend on conditions that hold during execution. Some other treatments of interference in the AI literature, such as Georgeff [Georgeff, 1986], provide for conditional interference, but only in the case when interference depends on conditions that hold just prior to execution.

We can detect whether *bai1* and *bai2* interfere at a world-history *h* by examining F_{cl} applied to *bai1* and *bai2* individually. Since we are assuming that *bai1*'s (and *bai2*'s) standard conditions hold everywhere, $F_{cl}(\{bai1\},h)$ yields a world-history in which *bai1* occurs. Call this world-history *h_z*. If *bai1* and *bai2* interfere at *h*, and consequently at *h_z*, $F_{cl}(\{bai2\},h_z)$ yields a world-history where *bai2* occurs (since its

standard conditions hold at h , but not $bai1$. If they do not interfere, both $bai1$ and $bai2$ occur in $F_{cl}(\{bai2\}, h)$ in which case we set $F_{cl}(\{bai1, bai2\}, h)$ to $F_{cl}(\{bai2\}, h)$.² Since $F_{cl}(\{bai2\}, h)$ is the result of modifying h first by $bai1$, then by $bai2$, it is equivalent to $bai1*bai2(h)$.

We can also detect if $bai1$ and $bai2$ interfere by modifying h first by $bai2$, then by $bai1$. $bai2*bai1(h)$ yields this world-history. If $bai1$ and $bai2$ interfere at h , then $bai1$, but not $bai2$, occurs in $bai2*bai1(h)$. If they do not interfere, both $bai1$ and $bai2$ occur in $bai2*bai1(h)$. Moreover, if they do not interfere, we assume that modifying h , first by $bai1$, then by $bai2$ yields the same world-history obtained by modifying h , first by $bai2$, then by $bai1$.

The definition of F_{cl} and constraints BA-CMP1 and BA-CMP2 capture the treatment described above. If $bai1$ and $bai2$ interfere with each other at h , then $bai1$ and $bai2$ do not both occur together in either $bai1*bai2(h)$ or $bai2*bai1(h)$. Consequently, $F_{cl}(\{bai1, bai2\}, h)$ is defined as $\{h\}$. If $bai1$ and $bai2$ do not interfere, then they occur together in both $bai1*bai2(h)$ and $bai2*bai1(h)$. In this case $F_{cl}(\{bai1, bai2\}, h)$ is set to $bai1*bai2(h)$ which equals $bai2*bai1(h)$ by constraint BA-CMP1. For the case where $bai1$'s and $bai2$'s standard conditions hold everywhere, constraint BA-CMP2 insures that $bai1*bai2(h)$ and $bai1*bai2(h)$ are compatible; they would be incompatible, if both $bai1$ and $bai2$ occurred together in one of them, signifying that $bai1$ and $bai2$ did not interfere at h , but did not occur together in the other, signifying they did interfere at h .

The analysis described above also applies in less restrictive cases where $bai1$'s and $bai2$'s standard conditions may not hold at all world-histories. This analysis is applicable as long as $bai1$'s standard conditions hold at both h and $F_{cl}(\{bai2\}, h)$, and $bai2$'s standard conditions hold at both h and $F_{cl}(\{bai1\}, h)$.

Let us now consider the case where both $bai1$'s and $bai2$'s standard conditions hold at h , but the occurrence of one of the basic action instances, say $bai1$, ruins the others standard conditions. This situation is treated as interference; $F_{cl}(\{bai1, bai2\}, h)$ is set to $\{h\}$. If $bai1$ ruins $bai2$'s standard conditions with respect to h then $bai2$'s standard conditions do not hold in $F_{cl}(\{bai1\}, h)$. Consequently, $bai1$, but not $bai2$, occurs in $bai1*bai2(h)$. By constraint BA-CMP2, both $bai1$ and $bai2$ will not occur together in $bai2*bai1(h)$ either. Thus, by the definition of F_{cl} , we see that $F_{cl}(\{bai1, bai2\}, h)$ is set to $\{h\}$.

The next case to consider is where $bai1$'s, but not $bai2$'s, standard conditions hold at h . In this situation, $F_{cl}(\{bai1, bai2\}, h)$ is set to $\{h\}$ unless the following two conditions hold: i) the occurrence of $bai1$ with respect to h brings about $bai2$'s standard conditions, and ii) they do not interfere with each other at h . If both i) and ii) hold, then both $bai1$ and $bai2$ occur together in $bai1*bai2(h)$. Consequently, by the definition of F_{cl} , $F_{cl}(\{bai1, bai2\}, h)$ is set to $bai1*bai2(h)$. This case differs from the situation where both $bai1$'s and $bai2$'s standard conditions hold

in h ; if $bai1$ brings about $bai2$'s standard conditions, then $bai1$ and $bai2$ do not necessarily occur together in $bai2*bai1(h)$ even though they do not interfere. Appropriately, constraints BA-CMP1 and BA-CMP2 are not applicable in this case.

The last case to consider is where both $bai1$'s and $bai2$'s standard conditions do not hold in h . In this situation, $F_{cl}(\{bai1, bai2\}, h)$ is simply set to $\{h\}$.

V. Conclusion

We have presented a model that provides for concurrent actions having temporal extent. We have integrated Allen's model [Allen, 1984], which can treat simultaneous events having temporal extent, with a structure analogous to the result function in situation calculus. This structure captures the result of executing an action at a specified time with respect to a context given by a world-history, i.e. a complete world over time. This enables us to model actions and action interactions that are affected by conditions that hold during execution. This structure also provides a simple framework for composing simple actions, both concurrent and sequential, to form complex ones.

Acknowledgments

The authors wish to thank Paul Benjamin for his comments on an earlier draft.

References

- [Allen, 1984] Allen, J.F., Towards a General Theory of Action and Time, *Artificial Intelligence* 23,2 (1984), 123-154.
- [Allen and Hayes, 1985] Allen, J.F. and Hayes P.J., A Common-Sense Theory of Time, *9th International Joint Conference on Artificial Intelligence*, Los Angeles, USA, August 1985.
- [Georgeff, 1986] Georgeff M., The Representation of Events in Multiagent Domains, *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, 70-75.
- [Goldman, 1970] Goldman A. I., *A Theory of Human Action*, Prentice Hall, Englewood Cliffs, NJ, 1970.
- [Haas, 1985] Haas A., Possible Events, Actual Events, and Robots, *Computational Intelligence* 1,2 (1985)
- [Lewis, 1973] Lewis D. K., *Counterfactuals*, Harvard University Press, Cambridge, MA, 1973.
- [McCarthy and Hayes, 1969] McCarthy J. and Hayes P., Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence*, vol. 4, Michie B.M.D. (ed.), 1969 463-502.
- [McDermott, 1982] McDermott D., A Temporal Logic for Reasoning about Process and Plans, *Cognitive Science* 6,2 (1982), 101-155.
- [Pelavin and Allen, 1986] Pelavin R.N. and Allen J.F., A Formal Logic of Plans in Temporally Rich Domains, *Proceedings of the IEEE* 17,10 (October 1986), 1364-1382.
- [Pelavin, 1987] Pelavin R.N., *A Formal Logic for Planning with Concurrent Actions and External Events*, PhD Thesis, University of Roch., 1987 (expected).
- [Stalnaker, 1968] Stalnaker R., A Theory of Conditionals, in *Studies in Logical Theory*, Rescher N. (ed.) Basil Blackwell, Oxford, 1968, 98-112.

² World-history h and the world-history in $F_{cl}(\{bai1\}, h)$ are not necessarily distinct from h . For example, if both $bai1$ and $bai2$ occur in h , then $F_{cl}(\{bai1\}, h) = F_{cl}(\{bai2\}, h) = \{h\}$ by constraint BA1; consequently $F_{cl}(\{bai1, bai2\}, h)$ equals $\{h\}$.

Preserving Consistency across Abstraction Mappings

Josh D. Tenenberg

Computer Science Department
University of Rochester
Rochester NY 14627

Abstract

An abstraction mapping over clausal form theories in first-order predicate calculus is presented that involves the renaming of predicate symbols. This renaming is not 1-1, in the sense that several predicate symbols $R_1, \dots, R_n$ from the original theory are all replaced by a single symbol R' in the abstract theory. In order to preserve consistency, however, the clauses that distinguish the R_i 's must be discarded in the abstract theory. This leads to a simple semantics; the union of the extensions of each of the R_i 's in any model of the original theory forms the extension of R' in a model of the abstract theory.

1 Introduction

An important method of constraining search in combinatorially intractable problems is to map the representation of the problem into an abstract representation, solve the problem in the abstract search space and use the abstract solution to guide the search in the original problem space. Work of this nature has been pursued by many researchers, most notably [Amarel 1972, Hobbs 1985, Korf 1985, Plaisted 1981, Sacerdoti 1974]. Of importance in this work is that certain formal properties hold between the original and the abstract versions of the problem in order to justify the use of abstraction. In this paper we will define a syntactic abstraction mapping between two first-order theories that involves the renaming of predicate symbols, a subclass of those mappings first defined in [Plaisted 1981]. The mapping is demonstrated to have the following properties. First, the abstract theory will be consistent if the original theory is; second, for each abstract theorem T' , there exists a theorem T in the original theory for which T' is an abstraction; third, the model-theoretic semantics of the abstract theory is intuitively appealing.

In section 2 we present the formal definition of a *predicate mapping*, and show in section 3 both its

relation to Plaisted's abstraction mappings, and its inherent problem of generating inconsistencies at the abstract level. Section 4 provides a set of syntactic restrictions that satisfies a particular semantics, and Section 5 demonstrates that these restrictions overcome the inconsistency problem by showing constructively that a model exists for the abstract theory given a model of the original theory. Section 6 will consist of a small example, and section 7 finishes with concluding remarks.

2 Predicate Mappings

We will assume standard first-order logic terminology as in [Enderton 1972]. In addition, all wffs will be taken to be in clause form [Robinson 1965]. Recall that a clause is a set of literals, and that clause C *subsumes* clause D if there exists a substitution ϕ such that $C\phi$ is a subset of D . In addition, C and D are *variants* if C and D are instances of one another; that is, they are identical except for a renaming of variables.

Predicate mappings are functions that map predicate symbols from one first order language to those of another. Given two sets of predicates P_1, P_2 ,

$$f: P_1 \rightarrow P_2,$$

where the P_i are the only symbols that possibly distinguish the first order languages L_1 and L_2 . What is noteworthy about f is that it is not necessarily 1-1, and therefore more than one relation symbol from L_1 can map to the same relation symbol from L_2 . We can then define f over literals such that literals in L_1 are mapped to literals in L_2 by replacing the predicate symbols under f . Likewise, f can be extended to clauses and sets of clauses in precisely this way.

3 Plaisted's Abstraction Mappings

Predicate mappings are a subclass of *abstraction mappings* that are defined in [Plaisted 1981]:

DEFINITION. An *abstraction* is an association of a set $f(C)$ of clauses with each clause C such that f has the following properties:

- (1) If clause C_3 is a resolvent of C_1 and C_2 and $D_3 \in f(C_3)$, then there exist $D_1 \in f(C_1)$ and $D_2 \in f(C_2)$ such that some resolvent of D_1 and D_2 subsumes D_3 .
- (2) $f(NIL) = \{NIL\}$. (NIL is the empty clause.)
- (3) If C_1 subsumes C_2 , then for every abstraction D_2 of C_2 there is an abstraction D_1 of C_1 such that D_1 subsumes D_2 .

If f is a mapping with these properties, then we call f an *abstraction mapping of clauses*.

This work was supported 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 Number F30602-85-C-0008 which supports the Northeast Artificial Intelligence Consortium (NAIC).

The author would like to thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

In addition, Plaisted proves the following theorem:

THEOREM. Suppose ϕ is a mapping from literals to literals. Let us extend ϕ to a mapping from clauses to clauses by $\phi(C) = \{\phi(L) : L \in C\}$. Suppose ϕ satisfies the following two properties:

- (1) $\phi(\neg L) = \neg(\phi(L))$. That is, ϕ preserves complements.
- (2) If C and D are clauses and D is an instance of C , then $\phi(D)$ is an instance of $\phi(C)$. That is, ϕ preserves instances.

Then ϕ is an abstraction mapping.

By this theorem, predicate mappings are abstraction mappings since they preserve both complement and instance.

Abstraction mappings have the property that from every clause C derivable from a set of clauses S , there exists a clause C' derivable from $f(S)$ that subsumes an abstraction of C (proven in [Plaisted 1981]). Informally, we might say that every solution in the original problem space has a corresponding solution in the abstract space. This will be termed the *upward-solution property*.

A problem with abstraction mappings, however, (and hence with predicate mappings) is that they may result in abstract theories which are inconsistent even though the original theory is consistent, which Plaisted termed the "false proof" problem. As an example, suppose we have a clause set for a simple domain in which we have objects that are bottles and glasses, with clauses stating that they are disjoint (note that all clauses will be in implication form for ease of interpretation):

$$1) \text{ Bottle}(x) \supset \neg \text{Glass}(x),$$

$$2) \text{ Glass}(x) \supset \neg \text{Bottle}(x),$$

and the clause stating that object A is a bottle:

$$3) \text{ Bottle}(A).$$

If our predicate mapping f maps both *Bottle* and *Glass* to a new predicate *Container* in the abstract theory, then from the abstract theory we can derive a contradiction:

$$\text{Container}(x) \supset \neg \text{Container}(x) \vdash \neg \text{Container}(A) \\ \text{Container}(A)$$

The approach taken in the remainder of the paper is to restrict the original set of clauses over which the predicate mapping can be applied so as to preserve consistency.

4 Restricted Predicate Mappings

The intuition behind the semantics of *restricted* predicate mappings is that we would like the interpretation of a predicate in the abstract theory to be the union of the interpretations of each of the predicates in the original theory that map to it. So the objects that are Containers are all of those objects that are either Bottles or Glasses, or any of the other things that map to Container.

We can obtain this interpretation syntactically by removing all of those axioms from the original theory that serve to distinguish the relations that are conflated at the abstract level. Relations $R_1, \dots, R_n$ in axiomatization θ_1 , can be made indistinguishable in axiomatization θ_2 by systematically replacing each R_i

with a new symbol R' under some predicate mapping f as before, but including in θ_2 only those clauses $f(C)$ such that C is in θ_1 and every clause mapping to C is derivable from θ_1 . So, for instance, if we wish to conflate the relations *Glass* and *Bottle* by mapping them to the same symbol, say *Container*, then given the axioms

$$\text{Glass}(x) \supset \text{Breakable}(x)$$

$$\text{Bottle}(x) \supset \text{Breakable}(x)$$

$$\text{Bottle}(x) \supset \text{Corkable}(x)$$

we would have in the new clause set only

$$\text{Container}(x) \supset \text{Breakable}(x),$$

since *Corkability* is true only of Bottles, and therefore distinguishes Bottles from Glasses in the original theory, while *Breakability* is true of both Bottles and Glasses. In other words, the clause

$$\text{Container}(x) \supset \text{Corkable}(x)$$

would not be placed in the new clause set since one of the clauses that maps to it, namely

$$\text{Glass}(x) \supset \text{Corkable}(x)$$

is not derivable from the original clause set.

This constraint, however, is stronger than required. For instance, it keeps one from ever asserting in the abstract theory that an object is a container in theories in which bottles and glasses are disjoint, as in (1)–(3) above, since it will never be the case that an object in the original theory is both a bottle and a glass. Given the desired semantics, it is permissible to allow the inclusion of mapped *positive* clauses from the original theory, those clauses containing no negated literals, into the abstract theory. That is, if object A has property P , then certainly object A will have the abstract property $f(P)$, and likewise for predicates of any arity. Thus, one can map the clause *Bottle*(A) to *Container*(A), since all bottles are containers, even though *Glass*(A) may not hold. In addition, this allows the mapping of clauses such as *Bottle*(A) $\vee$ *Glass*(A) to *Container*(A) $\vee$ *Container*(A).

5 Formal Definitions and Proof of Consistency

We use $[C]_f$ to denote the set of symbols or clauses that map into C under predicate mapping f . That is,

$$[C]_f = \{D \mid f(D) = C\}.$$

Each such D will be called a *specialization* of C . If θ is a clause set and D is a clause then we will use $\theta \vdash D$ to mean that the null clause can be derived from the clause set $\theta \cup \neg D$ by the inference rule *full resolution* [Robinson 1965]. Let θ be an axiomatization in clause form of a theory over language L_1 and let f be a predicate mapping from L_1 to L_2 . We define $g(\theta)$, a *restricted predicate mapping*, to be a subset of $f(\theta)$, where

$$(4) g(\theta) = \{C \mid \text{there exists some } D \in [C]_f \text{ such that } D \in \theta \\ \text{and either } C \text{ is a positive clause or} \\ \text{for every } D \in [C]_f \text{ it is the case that } \theta \vdash D\}.$$

The main theorem states that given a clause form axiomatization θ and a restricted predicate mapping g , $g(\theta)$ is consistent if θ is. In other words, consistency is preserved by the mapping. Given that a first-order clause set θ is consistent if and only if it is satisfiable, i.e., there exists a model that assigns truth to all of the clauses D such that $\theta \vdash D$, this theorem will be proven by constructing a model of $g(\theta)$ from a model of θ .

Standard formal semantic terminology will be assumed [Enderton 1972]. In particular, an *interpretation* is a

structure $\langle D, G \rangle$, where D is a set of objects, the domain, and G is an interpretation function that assigns to each constant symbol an object from D , to each n -function symbol an n -function mapping objects in D to an object in D , and to each n -predicate symbol an n -relation over objects in D . Further, we say that γ is a value assignment if it assigns to every variable symbol an object in D . If σ is a symbol from the language over which G is defined, we say that $G(\sigma)$ is the interpretation of σ under M , which in the case of an n -predicate will be a set of n -tuples. In addition, clauses are assigned truth values according to semantic rules pertaining to the logical symbols, which in the case of clause form is universal quantification, negation, and disjunction. If σ is a clause and γ is a value assignment, then $M(\sigma)$ returns either True or False under γ , denoted by $My(\sigma)$. If $My(\sigma) = \text{True}$ for every value assignment γ , then we say that σ is True in M , or $M(\sigma) = \text{True}$, otherwise σ is False in M . An interpretation $M = \langle D, G \rangle$ is a model of a clause set θ if D is True in M for every clause D such that $\theta \vdash D$.

Theorem:

Let θ be an axiomatization in clause form of a theory over language L_1 , let f be a predicate mapping from L_1 to L_2 , and let g be the restricted predicate mapping defined relative to f as in (4) above. For any model $M = \langle D, G \rangle$ of θ , $M' = \langle D, G' \rangle$ is a model of $g(\theta)$ where for every predicate symbol σ of L_2 ,

$$(5) \quad G'(\sigma) = \bigcup_{\xi \in \{\sigma\}} G(\xi).$$

and for every non-predicate symbol σ , $G'(\sigma) = G(\sigma)$.

Proof:

Let us assume that M' is not a model of $g(\theta)$. There must then be some clause $C \in g(\theta)$ that is False in M' . There are two cases to consider - first where C is a positive clause, and second where C contains negated literals.

Case 1: Assume C is a positive clause.

For membership of C in $g(\theta)$, it must be that there exists some clause $E \in [C]_f$ such that $E \in \theta$, by the definition of g . Since M is a model of θ , E is True in M . Then, for every value assignment γ , $My(E) = \text{True}$. Let us take truth in M and M' relative to a particular value assignment γ . E is of the form $E = \{P_1(a_1), \dots, P_n(a_n)\}$, where a_i represents the predicate P_i 's list of arguments. $My(E) = \text{True}$ means that $My(P_i(a_i)) = \text{True}$ for some literal of E . Since a_i represents the terms $(a_{i1}, \dots, a_{ij})$ of P_i , then $\langle G(a_{i1}), \dots, G(a_{ij}) \rangle \in G(P_i)$. Additionally it follows that $\langle G'(a_{i1}), \dots, G'(a_{ij}) \rangle \in G'(f(P_i))$ under γ , by the definitions of the interpretation function G' given in (5) above, and the predicate mapping f . That is, for $Q = f(P_i)$, $My(Q(a_{i1}, \dots, a_{ij})) = \text{True}$. But this literal is in C , and so $My(C) = \text{True}$. Therefore, $My(C) = \text{True}$ for every value assignment γ since γ was chosen arbitrarily, making $M'(C) = \text{True}$, contradicting our assumption. This dispatches the first case.

Case 2: Assume C contains negated literals.

For membership of C in $g(\theta)$, for every clause $E \in [C]_f$, $\theta \vdash E$ and some $E \in \theta$, by the definition of g . Since M is a model of θ , $M(E) = \text{True}$ for every such E , and therefore $My(E) = \text{True}$ for every value assignment γ .

Since C is False in M' , there must exist a value assignment γ such that $My(C) = \text{False}$. Let us take truth in M and M' relative to this value assignment γ .

C is of the form $C = \{R_1(a_1), \dots, R_n(a_n), \neg S_1(b_1), \dots, \neg S_k(b_k)\}$, where a_i and b_j are the arguments of each of the literals. That is, some of the literals in C are negated, and some may be positive. For every literal $L \in C$, $My(L) = \text{False}$. This means that for every $E \in [C]_f$, for every positive literal $W \in E$, $My(W) = \text{False}$, since were it not, then $My(g(W)) = \text{True}$ for some literal W and hence $My(C) = \text{True}$ by the definition of G' , contradicting our assumption that $My(C) = \text{False}$. Therefore, under this value assignment one or more of the negated literals in each $E \in [C]_f$ must be True.

Since each $\neg S_i(b_i)$ is False in M' under γ , then each $S_i(b_i)$ is True. That is, taking b_i to represent the ground terms $(b_{i1}, \dots, b_{ij})$ of each $\neg S_i$, $\langle G'(b_{i1}), \dots, G'(b_{ij}) \rangle \in G'(S_i)$ for each of the negated literals under γ . But by the definition of G' , for each of the S_i 's there exists some $Q_i \in [S_i]_f$ where $\langle G(b_{i1}), \dots, G(b_{ij}) \rangle \in G(Q_i)$ under this same value assignment. That is, $My(\neg Q_i(b_{i1})) = \text{False}$ for some specialization Q_i of each predicate S_i of each negated literal in C . Let E be the clause consisting of the negation of each such $Q_i(b_i)$ and some specialization of each of the positive literals in C . Note that $E \in [C]_f$, $My(E) = \text{False}$, since each of the literals in E are False. This then contradicts the earlier conclusion that every $E \in [C]_f$ be assigned True by M for every value assignment γ . This dispatches case 2, and hence establishes the theorem. QED

Restricted predicate mappings are no longer abstraction mappings by Plaisted's definition, since the upward-solution property is not preserved. However, abandoning the completeness afforded by the upward-solution property is likely what is required to solve frequently encountered problems at a lower cost than if they were solved directly in the original theory. These restricted mappings do have what can be termed the downward-solution property, since a trivial corollary of the theorem is that for every clause derivable from the abstract theory, there will be a specialization of it derivable from the original theory. Note that there may be no solution to the problem in the abstract theory since its solution requires some of the details that are ignored at that level. However, every abstract solution is guaranteed to have a specialized solution to the original problem.

An important objection that one might have is that the syntactic mapping function g given in (4) is undecidable, since it is based upon determining $\theta \vdash D$ for every clause D mapping to each candidate clause in the abstract clause set. One should note, however, that the search for derivability can always be arbitrarily bounded, and if no proof is obtained within this bound then it can be assumed that this clause is not derivable. In this way, consistency is still preserved between the original and the abstract theory, the abstract theory being simply weaker than it theoretically could be, i.e., having fewer theorems. The appropriate amount of resource that one should expend in constructing abstract theories is quite an important issue but outside the bounds of this paper. These kinds of performance choices, however, are discussed in [Hartman and Tenenberg 1987], elsewhere in this volume.

6 Example

Let θ , the original clause set, be the following clauses (note that $A \supset B$ is equivalent to $\neg A \vee B$, and is therefore not a positive clause):

- 1) $\text{Bottle}(x) \supset \text{MadeOfGlass}(x)$
- 2) $\text{Bottle}(x) \supset \text{Graspable}(x)$
- 3) $\text{Glass}(x) \supset \text{MadeOfGlass}(x)$
- 4) $\text{Glass}(x) \supset \text{Graspable}(x)$
- 5) $\text{Glass}(x) \supset \text{Open}(x)$
- 6) $\text{Box}(x) \supset \text{Graspable}(x)$
- 7) $\text{Bottle}(x) \supset \neg \text{Glass}(x)$
- 8) $\text{Glass}(x) \supset \neg \text{Bottle}(x)$
- 9) $\text{Bottle}(x) \supset \neg \text{Box}(x)$
- 10) $\text{Glass}(x) \supset \neg \text{Box}(x)$
- 11) $\text{Box}(x) \supset \neg \text{Glass}(x)$
- 12) $\text{Box}(x) \supset \neg \text{Bottle}(x)$
- 13) $\text{Bottle}(x) \supset \text{MilkBottle}(x) \vee \text{Winebottle}(x)$
- 14) $\text{Open}(x) \wedge \text{Graspable}(x) \supset \text{Pourable}(x)$
- 15) $\text{Graspable}(x) \supset \text{Movable}(x)$
- 16) $\text{MadeOfGlass}(x) \supset \text{Breakable}(x)$
- 17) $\text{Bottle}(x) \vee \text{Glass}(x) \vee \text{Box}(x)$
- 18) $\text{Open}(A)$

Let f_1 map each predicate symbol to itself except that it maps *Bottle* and *Glass* both to *GlassContainer*. Therefore, $g_1(\theta)$ is the following:

- 1') $\text{GlassContainer}(x) \supset \text{MadeOfGlass}(x)$
- 2') $\text{GlassContainer}(x) \supset \text{Graspable}(x)$
- 6') $\text{Box}(x) \supset \text{Graspable}(x)$
- 9') $\text{GlassContainer}(x) \supset \neg \text{Box}(x)$
- 11') $\text{Box}(x) \supset \neg \text{GlassContainer}(x)$
- 14') $\text{Open}(x) \wedge \text{Graspable}(x) \supset \text{Pourable}(x)$
- 15') $\text{Graspable}(x) \supset \text{Movable}(x)$
- 16') $\text{MadeOfGlass}(x) \supset \text{Breakable}(x)$
- 17') $\text{GlassContainer}(x) \vee \text{Box}(x)$
- 18') $\text{Open}(A)$

Note that clauses 3 and 4 are redundant in the abstract theory and are therefore not included in $g_1(\theta)$, as are 10 and 12. Clause 5 gets eliminated because $\text{Bottle}(x) \supset \text{Open}(x)$ is not derivable from θ . Likewise with 7 and 8 as seen earlier. Additionally, 17' was factored so that the same literal did not appear twice. Although $g_1(\theta)$ is much smaller than θ , there are still several intuitive inferences that can be made, such as $\text{Graspable}(x)$, $\text{Movable}(x)$ and $\text{Pourable}(A)$.

Given the way in which abstraction is defined, $g_1(\theta)$ can additionally be treated as a primitive level theory and mapped into a more abstract theory $g_2(g_1(\theta))$. For instance, let f_2 map all predicates to themselves except *Box* and *GlassContainer* get mapped to *Container*. Thus, $g_2(g_1(\theta))$ is:

- 2'') $\text{Container}(x) \supset \text{Graspable}(x)$
- 14'') $\text{Open}(x) \wedge \text{Graspable}(x) \supset \text{Pourable}(x)$
- 15'') $\text{Graspable}(x) \supset \text{Movable}(x)$
- 16'') $\text{MadeOfGlass}(x) \supset \text{Breakable}(x)$
- 17'') $\text{Container}(x)$
- 18'') $\text{Open}(A)$

One can therefore abstract a theory arbitrarily many times, preserving consistency throughout. Each abstract theory has the advantage that those clauses that it can derive will likely be done so at considerably less cost than in the original theory. Although this would be true of any subset of θ , it is important to recognize that the mapping functions embed a significant amount of heuristic information about the domain. This function determines which clauses are important, and which are mere details, and so the

choice about which clauses remain in the abstract theory is not done so on an arbitrary basis.

7 Conclusion

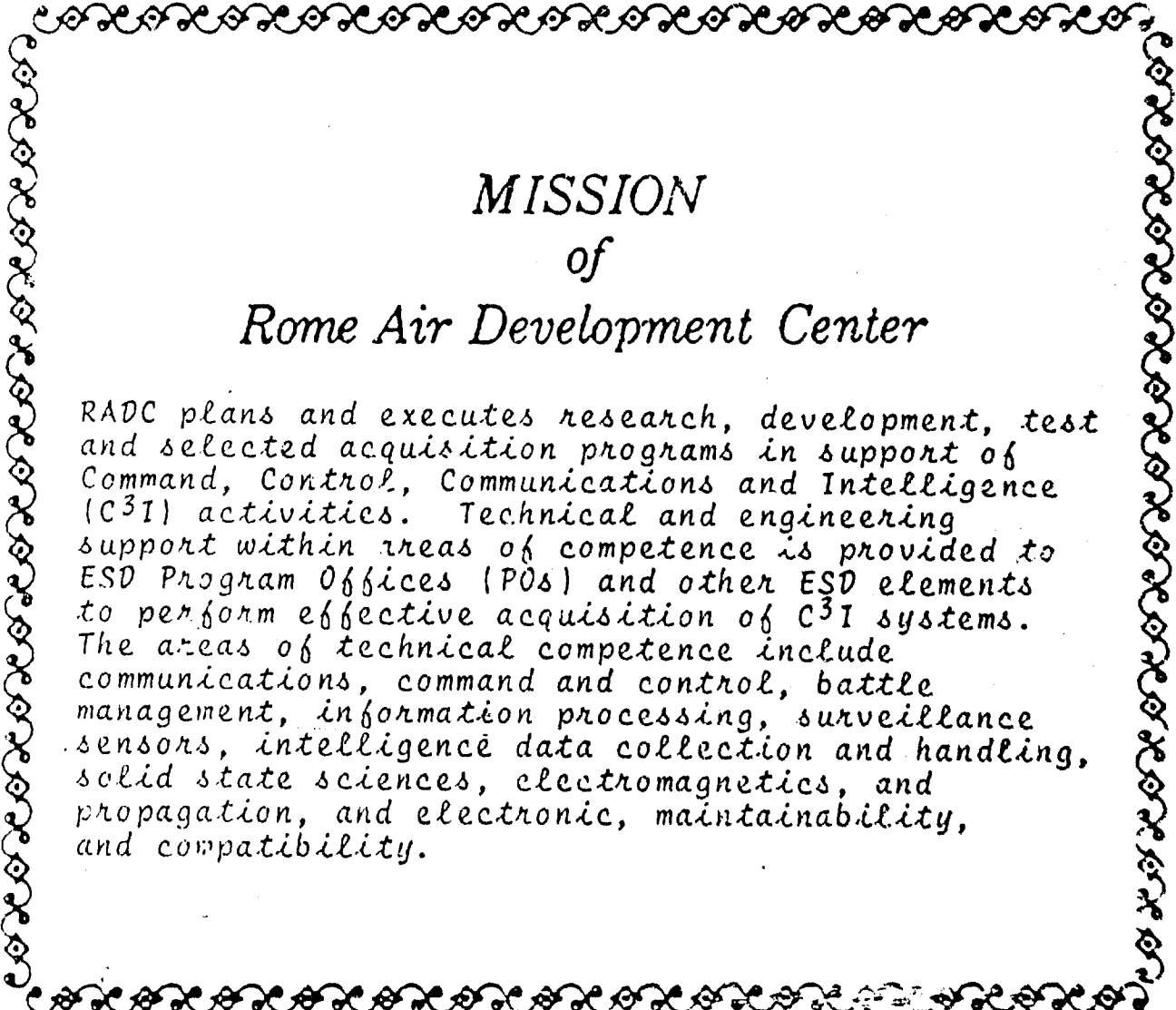
A useful way of viewing abstraction is as a function that maps one first-order theory into another. We have defined *predicate mappings* that involve the renaming of predicate symbols, where more than one of the original symbols can map into the same abstract symbol. These predicate mappings are a subclass of Plaisted's abstraction mappings, and as such suffer from the fact that consistency in the original theory may be lost in the abstract theory. To remedy this, we introduced syntactic restrictions to the predicate mapping that guarantee the preservation of consistency. These restrictions amount to eliminating the axioms in the original theory that distinguish the predicates which are being conflated in the abstract theory. This has a simple semantics; the extension of each of the predicates P in a model of the abstract theory can be constructed from the union of the extensions of each of the predicates that map to P in any model of the original theory. A proof of this was provided, along with an example demonstrating one possible use of the restricted predicate mappings.

Acknowledgements

I would like to thank my advisor, Dana Ballard, for his encouragement and wealth of ideas on abstraction. In addition, I would like to thank members of the UR-AI group, especially Leo Hartman, who always seems a step ahead, sometimes two, and Jay Weber, for pushing for a semantic definition. I am also grateful to Mark Fulk, for finding the problems with the old versions of the main proof, and pointing the way to the correct version.

References

- Amarel, S., "On representations of problems of reasoning about actions", In D. Michie (editor), *Machine Intelligence 3*, 131 - 171. American Elsevier, New York, 1972.
- Enderton, H., *A Mathematical Introduction to Logic*, Academic Press, Inc., London, 1972.
- Hartman, L., and Tenenberg, J., "Performance in Practical Problem Solving", *Proceedings - IJCAI*, Milan, Italy, 1987.
- Hobbs, J., "Granularity.", *Proceedings - IJCAI*, Los Angeles, CA 1985.
- Korf, R., "An analysis of abstraction in problem solving", *Proceedings - ACM Technical Symposium on Intelligent Systems*, 1985.
- Plaisted, D., "Theorem Proving with Abstraction", *Artificial Intelligence* 16:47-108, 1981.
- Robinson, J., "A machine-oriented logic based on the resolution principle", *Journal of the ACM* 12:23-41, 1965.
- Sacerdoti, E., "Planning in a hierarchy of abstraction spaces", *Artificial Intelligence* 5:115 - 135, 1974.



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 (C³I) 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 C³I 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, maintainability, and compatibility.