

AD-A182 724

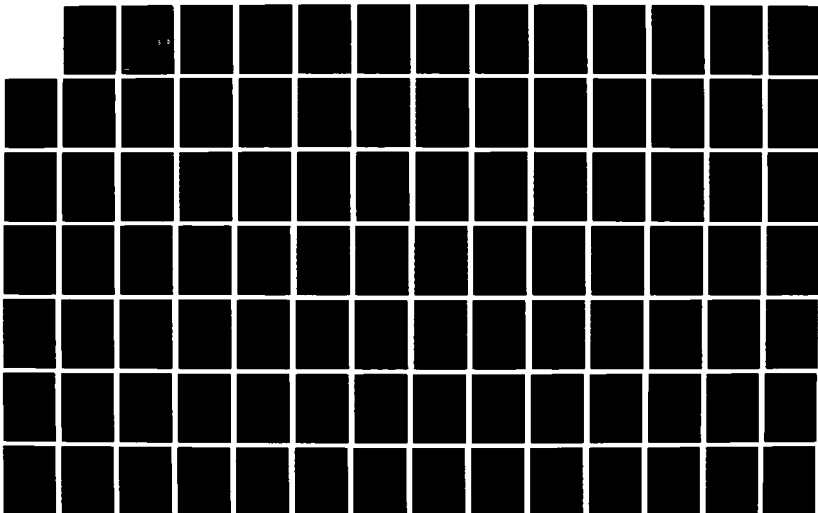
ONR (OFFICE OF NAVAL RESEARCH) RESEARCH IN DISTRIBUTED
REASONING AND PLANNING(U) SRI INTERNATIONAL MENLO PARK
CA ARTIFICIAL INTELLIGENCE CENTE. K G KONOLIGE ET AL.
MAY 87 N00014-85-C-0251

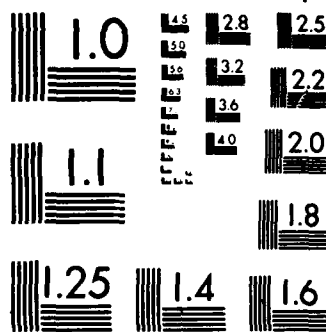
1/2

UNCLASSIFIED

F/G 12/7

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A182 724

ONR RESEARCH IN DISTRIBUTED REASONING AND PLANNING

INTERIM REPORT

Covering Period - February 1985 to February 1987

May 1987

By: Kurt G. Konolige, Computer Scientist
Michael P. Georgeff, Program Director
Amy L. Lansky, Computer Scientist

Representation and Reasoning Program
Artificial Intelligence Center

Prepared for:

Office of Naval Research
800 North Quincy
Arlington, Virginia 22217
ATTN: Dr. Alan Meyrowitz

ONR Contract No. N00014-85-C-0251
SRI Project 8342

APPROVED

Michael P. Georgeff, Program Director
Representation and Reasoning Program

C. Ray Perrault, Interim Director
Artificial Intelligence Center Computer and Information Sciences Division

DTIC
ELECTE
JUL 20 1987
S D

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited



INTRODUCTION

The goal of the Distributed Reasoning and Planning projects has been to provide fundamental research into the principles upon which a network of intelligent, cooperative computer agents can be based. In this Interim Report we describe work conducted by SRI under Contract N00014-85-C-0251 with the Office of Naval Research in the two years ending February, 1987.

Background

The motivation for distributed systems of intelligent agents arises when we consider problem domains with the following characteristics:

- The environment is naturally partitioned, either spatially or functionally. An example of the former is a surveillance task in which several different areas must be patrolled; of the latter, assembly tasks in which major subcomponents can be assembled in parallel.
- Each process's view of the environment is dynamic: other processes may be performing tasks at the same time, and chains of events may result from ongoing natural processes.
- Each process has partial or imperfect knowledge of the environment.
- The cost of communication among processes may be high or the communication channels may be slow, so that there is a significant delay in sending large amounts of information.

This multiagent, dynamic domain is very different from the static, single-agent domains of much previous planning research in artificial intelligence. Because the cost of communication is high, each process reacts intelligently on its own to changes in the environment. Furthermore, because of the uncertainty inherent in information gathered from sensory apparatus and limitations on the functional capabilities of the processes, each process must have a well-developed model of its environment (including the structure of other processes), and the ability to reason about actions and events in quite complex ways. We call such autonomous intelligent processes *agents*. The reasoning abilities needed by these agents fall into the following general categories:

- Reasoning about the environment on the basis of what is known, including reasoning about the beliefs and intentions of other agents.
- Communicating to exchange information about the environment and intentions to act.
- Reasoning about actions and events, including reasoning about the effects and interaction of actions.
- Forming cooperative plans on the basis of this information.
- Monitoring and synchronizing the execution of individual plans.
- Using explicit information about the structure of the domain (the various locations and their structural interrelationships) to help guide the formation and synchronization of the plans of multiple agents.

These issues have been at the core of our research efforts in distributed reasoning and planning over the past five years. This research has naturally fallen into two parallel, yet interrelated, paths. The first path has primarily involved the search for a solution to the technical problems of coordinating the plans and behaviors of multiple agents. Research along this path has assumed that the types of goals to be achieved have been defined and that the task at hand is to generate synchronized plans for achieving those goals. This closely resembles the traditional notion of planning in AI, and the natural starting point for our research was to look at existing single-agent planners to see whether they could be extended to accommodate multiagent worlds. Over time, however, we discovered that the underlying models of action for single-agent planning were not sufficiently rich to capture the complicated synchronization properties of multiagent worlds. Subsequent research then began to explore richer action representations, in particular to make use of work that had been done in computer science concurrency theory.

Our second research path has focused on the problem of designing agents that are both *autonomous* and capable of performing cooperative tasks in multiagent environments. Our basic strategy was to examine carefully how humans actually function—in essence, to model agents as entities with explicit beliefs, desires, and intentions. These cognitive components of human “rationality” were examined and formalized, and realistic computational techniques were formed to reason with them. This work is related to the first research path in that it forms a basis for the actual selection of goals. It also directly affects the planning process, because agents’ beliefs about one another can affect how they plan and synchronize with each other.

Out of this second line of research has grown another focus of our attention: providing an account of reasoning about defaults. This research is essential to the development of distributed planning agents, because such agents will normally have only partial information about one another’s plans. Given the cost of communication, these agents must be able to rely on default reasoning to fill in missing information in a plausible way.

We have been conducting research in each of these areas on this project, and have achieved significant progress. We briefly review this work below.

CURRENT RESEARCH RESULTS

Representing and Reasoning About Cognitive States

One of the major research issues that we studied has been the use of concepts from symbolic logic and theoretical computer science to rigorously characterize the notion of a rational cognitive agent. In particular, we investigated the role of knowledge, belief, desire, intention, planning, and action from several points of view, as follows.

- Their formal properties as studied in idealized models abstracted from common sense
- Their respective roles in allowing an agent to carry out complex purposive behavior
- Various computational realizations.

Under our intuitive model of intelligence, an agent chooses actions by foreseeing their consequences and measuring these against his goals. In the case of multiple agents, part of the problem of projecting consequences involves reasoning about how an action can affect what other similar agents are likely to believe, want, and do. While this commonsense view of intelligence has served as the implicit basis of much work in distributed AI planning, very little had been done previously to formalize this concept in a way that could serve as the basis for concrete planning algorithms. Indeed, the formalization of the key elements of cognitive state still remains a significant challenge.

Kurt Konolige's thesis [51] made significant progress in formalizing belief states. Previous formal models of belief were derivatives of a possible-world semantics for belief. However, this model suffers from epistemological and heuristic inadequacies. Epistemologically, it assumes that agents know all the consequences of their belief. This assumption is clearly inaccurate, because it doesn't take into account resource limitations on an agent's reasoning ability. It is also computationally inefficient.

A more natural model of belief is Konolige's deduction model: an agent has a set of initial beliefs about the world in some internal language, and a deduction process for deriving some (but not necessarily all) logical consequences of these beliefs. Within this model, it is possible to account for resource limitations of an agent's deduction process; for example,

For	
Ed	☒
S	☐
ed	☐
A-1	
CHI	Acad. Prod. for Special

one can model a situation in which an agent knows the rules of chess but does not have the computational resources to search the complete game tree before making a move.

In the beginning years of our work, we investigated a Gentzen-type formalization of the deductive model of belief, and have proved several important and original results. Among these are soundness and completeness theorems for a deductive belief logic; a correspondence result that shows the possible-worlds model is a special case of the deduction model; and a modal analog to Herbrand's Theorem for the belief logic. Several other topics of knowledge and belief have been explored from the viewpoint of the deduction model, including a theory of introspection about self-beliefs, and a theory of circumscriptive ignorance, in which facts an agent *doesn't* know are formalized by limiting or circumscribing the information available to him.

A major part of our research effort in the last two years has been the computational implementation of the deduction model of belief. We have completed proofs of soundness and completeness for resolution in a modal logic of belief, as described in Chapter 1. These methods are significant and unique for a number of reasons. They are the first practical *direct* theorem-proving methods for a modal logic, that is, that do not rely on a translation of the modal language into a first-order one. Also, they use the notion of *semantic attachment* as an efficient means for reasoning about the beliefs of other agents. Basically, this involves setting up a model of the agent's belief derivation process, and actually running this process to determine what beliefs that agent has. A preliminary implementation, built in collaboration with a graduate student, Christophe Geissler, has successfully solved some complicated puzzles that are testbed problems for epistemic theories. These results are reported in Chapter 2.

Currently we are integrating our belief resolution methods with a state-of-the-art theorem prover using connection graph methods. We intend to use the resulting system to represent and reason about concurrent actions, "spontaneous" events outside the immediate control of the planner/executor, and the effects of an agent's actions on the beliefs, goals, and plans of other agents.

In other work, we have been developing a theory of default reasoning that relies on epistemic concepts, and so can be integrated with our theory of deductive belief. The particular kinds of defaults we are interested in are expressed by statements such as "Typically, A's are B's." In the absence of any contradictory information, we wish to conclude that any particular A is also a B, *e.g.*, from "Typically birds fly" and "Tweety is a bird" we should infer that Tweety flies. But this conclusion can be denied if there is information which defeats the application of the default; we may know that Tweety is a penguin, for example. We have used a theory of default reasoning that distinguishes several different types of defeat for default rules, and introduced by Pollock [93]. In collaboration with another graduate student, Karen Myers, we have developed this theory further, and shown how to incorporate various intuitive constraints on defaults into a formal epistemic system. These results are in Chapter 3.

Reasoning About Action

The classic STRIPS approach to planning [25] represents the effects of actions using operator descriptions that assume a single-agent world. Our initial approach to multiagent planning was to extend some of the STRIPS ideas to allow the possibility of multiple agents. One technique treats actions as *atomic* events, where the actions of multiple agents are interleaved in sequence. This technique was found to be inherently insufficient, because it does not account for the possibility of simultaneous activity and its consequent effects. Turning to recent work in concurrency theory, we next attempted to blend synchronization techniques with traditional state-based planning.

One direction of research concerned the question of how distributed problem solvers could plan to perform distinct tasks without interfering with one another, while allowing cooperation where necessary. We envisioned a number of agents, each *separately* forming plans to accomplish given tasks, and then, by communicating with one another or a central scheduler, modifying their plans to avoid interference with one another. These plans, though separate, could include cooperative actions that are to be performed during a single interval of time.

Battle management provides a typical scenario, where individual commanders might form their own plans and then coordinate these with one another. A different scenario would be a multicrew aircraft, where, for example, one crew member might form plans for diagnosing an engine malfunction while other crew members separately form plans to complete the mission or to analyze incoming radar information. At some stage in the process, these plans must be coordinated so that the execution of one task does not interfere with the execution of others.

It is assumed that, having formed the separate plans, the individual agents communicate information about these plans to one another, or to a central scheduler, by describing the possible sequences of actions they intend to carry out. In our previous research, we were concerned with describing means by which a central scheduler could analyze the plans and advise the individual agents how to coordinate them. We did not want the scheduler to impose unnecessary constraints on the order in which actions were performed: we wanted the individual agents to have maximum flexibility in executing their own tasks and not be required to wait for other agents unless it was necessary to avoid interference or to allow for cooperation on some subtask.

Although we have assumed use of a central scheduler, the problem-solving methods could also be distributed to a number of agents. Furthermore, we can, although we need not, assume that the agents know enough about one another that they include in their plans only those actions that are observable by other agents. Internal actions of a given agent can be safely excluded from the plans that are communicated to the central scheduler.

A central scheduler must first be able to determine, from descriptions of the actions occurring in the plans, which actions may interfere with one another, and in what way. For example, it may be that two actions cannot be performed at the same time, or that one action must be begun before another is finished. This requires that we have a model of action and an action description language that allows us to determine potential interference.

Having determined which actions can interfere with one another, and the way in which they interfere, the central scheduler must construct a coordinated plan that avoids this interference. To do this, it must determine which possible orderings of actions satisfy the constraints of noninterference. Having made that determination, it can then insert synchronization actions—interagent communications—into the original plans to ensure that only these interference-free orderings are allowed. For maximum flexibility, we require that the synchronization be such that *all* potential orderings can be generated, rather than impose additional and unnecessary constraints on the behavior of the individual agents.

In performing this work, we first developed a more powerful representation of action than that used by traditional planners [28]. We represent actions as sequences of states, rather than as simple state-change operators. This approach allows the expression of more complex kinds of interaction than would otherwise be possible. The action representation was eventually enhanced to include sequencing, selection, nondeterminism, iteration, and parallelism; it is thus much more general than most models of action. It also provides a basis for understanding and reasoning about action sentences in both natural and programming languages.

Next, an efficient method of interaction and safety analysis was developed and used to identify critical regions in the plans [28]. An essential feature of the method is that the analysis is performed without generating all possible interleavings of the plans, thus avoiding a combinatorial explosion. The original approach, developed by Michael Georgeff, achieved this when the plans of the individual agents were linear. He showed the importance of the STRIPS assumption in substantially reducing combinatorics, and how to use a commutativity law to further reduce combinatorics. In subsequent work [29], he showed how to determine freedom from interference for arbitrary—rather than linear—plans, but did not show how, if there was interference, such plans could be coordinated.

These ideas were used by Chris Stuart [111], who implemented a synchronizer based on techniques developed by Wolper and Manna [74]. Whereas Georgeff's techniques could only find areas of interference between single-agent plans, Stuart's synchronizer can solve for more arbitrary global temporal constraints. However, these constraints are restricted to being propositional: they must refer to existing actions. Thus, synchronization that involves the creation and addition of new actions cannot be accomplished with Stuart's system.

In the past year, our research has focused on departing from traditional state-based views, favoring instead an event-based representation of domain. In part, this departure resulted from our experiences with constructing the synchronizer described above, as we usually found ourselves thinking more in terms of events than of the states affected by events. Some of our work has involved relating state-based and event-based reasoning in an essentially state-based framework [30,31], which, unlike previous state-based models of action, provides for simultaneous action. We constructed a model-based law of persistence to describe how actions affect the world. No frame axioms or syntactic frame rules are involved in the specification of any given action, thus allowing a proper model-theoretic semantics for the representation. We identified some serious deficiencies in existing approaches to reasoning about multiple agents. Finally, we showed how the law of persistence, together with a notion of causality, makes it possible to retain a simple model of action while avoiding

most of the difficulties associated with the frame problem. This work is described in Chapter 4.

Our other approach was to depart completely from state-based models and represent the world directly in terms of structured, interrelated events [63,64,61]. The basis of this work is GEM (the Group Element Model). A new multiagent planner based on this representation, GEMPLAN, is currently being developed. Besides being based on the very different, event-based approach, GEMPLAN will accomodate much broader forms of plan synchronization than Georgeff and Stuart's planners, including the satisfaction of a subset of first-order temporal logic constraints. Chapter 5 describes our research in this area.

As discussed earlier, the interest in underlying principles of rationality, both for planning and for interpreting the actions and utterances of others, is an important characteristic of our work in distributed reasoning. In particular, these principles are often thought to include design principles for agents with limited resources who must cope with a changing environment in real-time. To this end, we have developed a reactive reasoning system, called a *Procedural Reasoning System* (PRS), that can reason about and perform complex tasks in dynamic environments. It has been used for performing the malfunction handling tasks as used on NASA's space shuttle and for the control of the new SRI robot, Flakey, to accomplish a series of tasks on a simulated space station.

To allow for performing multiple reasoning tasks concurrently (either by a single agent or by a number of different agents), PRS has been designed so that several instantiations of the basic system can run in parallel. Each PRS instantiation has its own beliefs, goals, and plans, and operates asynchronously relative to other PRS instantiations, communicating with them by sending messages. The messages are written into the belief data base of the receiving PRS, which must then decide what to do, if anything, with the new information. As a rule, this decision is made by a fact-invoked plan of action (in the receiving PRS), which responds upon receipt of the external message. In accordance with such factors as the reliability of the sender, the type of message, and the beliefs, goals, and current intentions of the receiver, it is determined what to do about the message – for example, to acquire a new belief, establish a new goal, or modify intentions. It remains to investigate just what kinds of communication are best suited to what environments. However, the message-passing mechanisms we have employed should allow us to integrate more complex reasoning about interprocess communication, such as that developed by Cohen and Levesque [16], as described in Chapter 6.

FUTURE PLANS

In the next two years, we plan to concentrate our research on representing and reasoning about cognitive state. Our current results on the representation of belief, and computational implementations of formal systems for reasoning about it, have proven to be exciting, and we wish to extend them to incorporate reasoning about the intentions, plans and goals of other agents. As we have argued above, this is an important and critical development for building truly autonomous agents. While there has been some AI research on representing and reasoning about the plans of other agents, especially for plan inference (inferring the plans of agents from their actions; see, for example, [4,15,71]), this work has generally ignored the complex interaction that exists among beliefs, intentions, and plans. Two exceptions are the recent thesis of Martha Pollack on inferring the plans of agents whose beliefs may be mistaken [?], and the work of Cohen and Levesque [14], in which the interaction of belief, intention and goals are formalized using a possible-worlds semantics. Martha Pollack will be joining the project, and we intend to pursue the development of a general representational framework for plans, intentions, and belief. As in the development of the deduction model of belief, the methodology we use will be experimental robot psychology (see Konolige [52]), that is, we will examine the design of a typical robot planning agent, and abstract the essential properties of its cognitive structure, concentrating on the relationship between its plans and beliefs about the world. We will try to avoid the pitfall of over-simplifying assumptions: for example, we will not suppose that the robot agent's beliefs are correct, nor that its actions will always succeed.

Once we have the abstract model specified, we will then formalize it as a means of reasoning about the agent. Here we will be guided by our experience with the deduction model of belief, and our recent work on defaults (Chapter 3). By using default rules, it is possible to state simply what is normally or typically the case, while still allowing for the possibility of complications. This is especially congenial for the formalization of the interaction of intention and belief, where the normal case can be described in a straightforward manner, but taking into account the various ways in which things can go wrong causes unacceptable complications in the formalization process.

Chapter 1

RESOLUTION AND QUANTIFIED EPISTEMIC LOGICS

This work was reported in the Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, California, 1986. It is written by Kurt Konolige.

A Introduction

Quantified modal logics (QML) have emerged as an important tool for reasoning about knowledge and belief in Artificial Intelligence (AI) systems. The idea of formalizing the basic properties of knowledge and belief in QML originated with Hintikka [44], who was interested in the analysis of several epistemic paradoxes. Subsequently he reformulated the semantics of his work using Kripke's notion of relative accessibility between possible worlds [45]. In the computer science community, McCarthy [77], Sato [101], Moore [84], Levesque [67], Halpern and Moses [38] and others have used variations of his approach to formalize and reason about knowledge and belief.

Whether quantified modal logics of this sort are appropriate as *epistemic* logics is controversial, both in philosophy and AI. The major objection is that they assume agents are perfect reasoners, so that they know all the logical consequences of their knowledge. Several attempts have been made to modify the possible-world semantics to avoid this assumption [68,23], and there are also other formal approaches which take into account the limited reasoning power of agents (for example, [51]). It is not the purpose of this paper to comment on the relative merits of these approaches; quantified modal logics with Kripke semantics are an important research tool for epistemic reasoning in computer science at present, and will probably remain so. Here we will be concerned with proof methods for these logics that could be used in automatic deduction systems. Surprisingly, there has been relatively little work in this area, although decision procedures for the propositional case have been explored (see Halpern and Moses [39]).

In this paper we lay the theoretical groundwork leading to the derivation of a resolution procedure for certain quantified modal logics. The procedure has been implemented and successfully solves a version of a standard benchmark in epistemic reasoning, the Wise Man Puzzle [27]. In outline, the derivation is as follows.

Kripke [56] has given a completeness proof for quantified $S5$, based on the analytic tableaux method. We build on his results, extending them in two main areas. First, Kripke's methods work only with constants which have a fixed interpretation relative to a single domain (*rigid designators*). For expressivity it is important to have terms whose interpretation varies over possible worlds; it is also technically necessary for the development of resolution methods, since skolem functions, by their very nature, cannot be rigid designators. However, there are well-known problems with the substitution of nonrigid terms into modal contexts. We solve these by the technical device a *bullet constructor*, essentially a method for turning nonrigid terms into rigid ones in the proper context.

Second, we isolate a concept, that of reducing the unsatisfiability of a set of modal atoms to the unsatisfiability of some set of their arguments, that is the key step in proving completeness. We will show how any system which possesses a reduction theorem of the proper form is complete.

Third, using a method from Smullyan [107], we put analytic tableaux for prenex sentences into a form in which all operations on quantifiers precede those on truth-functional connectives. Then, as a corollary to the completeness proof, we have a version of the Skolem-Herbrand-Gödel theorem for modal systems: a sentence is unsatisfiable if and only if a finite conjunction of its instances is. This theorem is the foundation of all automatic deduction procedures for first-order logic, including Robinson's resolution method [96].

With the help of the bullet constructor, it is possible to eliminate existential operators from a sentence in prenex normal form, a process referred to as *skolemization* in first-order systems. Finally, using our Skolem-Herbrand-Gödel theorem and drawing on the technique of theory resolution (from Stickel [110]), we show how the reduction theorem for a modal system leads to a sound and complete resolution system.

Because of space limitations, it is impossible to give proofs of the theorems in this paper.

B Logical preliminaries

B.1 Epistemic logics

We consider six quantified modal logics that are typically used in reasoning about knowledge and belief (see Halpern and Moses [39]); we call these collectively *epistemic logics*. All of the logics have the following properties. Their language is first-order with the addition of a modal operator of the form $B\phi$, where ϕ is a formula of the language. Informally, $B\phi$ means that the agent believes or knows the proposition expressed by ϕ . All of the results of this paper are easily extended to the case where there is a sequence of modal operators B_i indexed by agent, but for simplicity we present the single-agent case.

Both arbitrary nesting of operators and "quantifying in" (i.e., statements of the form $\exists x.B\phi(x)$ or $\forall x.B\phi(x)$) are allowed in the language. In addition, there is a bullet construction $\bullet t$, where t is a term not containing any bullet operators. A *sentence* is a formula which has no free variables, and whose bullet constructions are all under the scope of a modal operator. A *modal atom* is a formula $B\phi$; if ϕ contains no variables, it is a *ground modal atom*. A *modal literal* is either a modal atom or its negation.

We will use uppercase Greek letters (Γ, Δ , etc.) to stand for denumerable sets of formulas; if $\Gamma = \gamma_1, \gamma_2, \dots$, then $B\Gamma$ abbreviates $B\gamma_1, B\gamma_2, \dots$, and $\neg B\Gamma$ abbreviates $\neg B\gamma_1, \neg B\gamma_2, \dots$.

C Semantics

The semantics of these logics is the standard Kripke possible-worlds model. A *frame* is a structure $\langle W, R \rangle$, where W is a set of possible worlds, and R is a binary relation on W . A particular logic will often place restrictions on the type of relation allowed in frames. e.g., in some epistemic logics (see below) R is transitive. In this paper we will restrict ourselves to a single relation for simplicity; the generalization to families of operators is straightforward.

A *model* consists of a frame, a special world $w_0 \in W$ (the *actual world*), a domain D_i for each world $w_i \in W$, and a valuation function V . At each possible world, V assigns a value to each term and sentence of the language. V obeys first-order truth-recursion rules: it also obeys particular rules for the modal operators, depending on the logic.

If $V(w, \phi) = \text{true}$, then we write $\models_w^w \phi$. $\models_m \phi$ is an abbreviation for $\models_m^{w_0} \phi$. If ϕ is true in all models of a logic A , we write $\models_A \phi$ or simply $\models \phi$ if the logic is understood.

The bullet construction has a special semantics. No matter where it occurs in a formula, $\bullet t$ always refers to the actual individual denoted by t , so that for all $w \in W$, $V(w, \bullet t) = V(w_0, t)$.

Different constraints on R yield different versions of epistemic logic. We consider the following variations:

Logic	Restriction on R
K	none
$K4$	transitive
$K45$	transitive, euclidean
T	reflexive
$S4$	reflexive, transitive
$S5$	equivalence

The first three logics ($K, K4, K45$) have belief as their intended interpretation. K is the simplest of these, placing the fewest restrictions on beliefs. $K4$ and $K45$ represent various types of introspective properties. In $K4$, if one believes something, one believes one

believes it ($B\phi \supset BB\phi$). $K45$ has this and its converse: if one doesn't believe something, one believes one doesn't believe it ($\neg B\phi \supset B\neg B\phi$).

The three logics which have reflexive R are logics of knowledge. The distinguishing characteristic here is that knowledge must be true ($B\phi \supset \phi$). T , $S4$ and $S5$ are the epistemic logics corresponding to K , $K4$ and $K45$.

It must be stressed that the purpose of the paper is not to argue for the appropriateness of these logics for modeling epistemic concepts. Indeed, it is easy to find problems here; for example, there are good reasons for denying that knowledge is only *true* belief, since it also seems to involve some complex notion of justification; and this is not formalized in T , $S4$, or $S5$.

Truth-recursion equations for these logics are the same. Along with rules for the boolean operators and quantifiers, we add the following rule for the modal operators:

$$V(w, B\phi) = \text{true} \quad \text{iff} \quad \forall w'. wRw' \rightarrow V(w', \phi) = \text{true} \quad (1.1)$$

C.1 Substitution

Substitution of terms for quantified-in variables is problematic, since it does not preserve validity. Consider the following example of an agent's beliefs.

$$\begin{aligned} &P(m(c)) \\ &\neg BP(m(c)) \\ &\forall x. Px \supset BPx \end{aligned} \quad (1.2)$$

We can construct a model as follows. Let P be the property of being non-Italian, let $m(x)$ denote the mayor of the city x , and c denote New York. Suppose the agent believes the mayor of New York is Fiorello LaGuardia (and not Ed Koch, the actual mayor); it is easy to confirm that all the sentences are satisfied.

Now if we substitute $m(c)$ for x in the third sentence, the resulting set is unsatisfiable. The reason is that, although x must refer to the same individual in all possible worlds, the substituted expression $m(c)$ need not. So even if a universal sentence is true in a model, some of its instances can be false.

Our solution to this problem is to redefine the meaning of "instance" by introducing a bullet construction ($\bullet$) whenever there is a substitution for variables inside the context of modal operators. In the above example, substituting $m(c)$ for x yields

$$P(m(c)) \supset BP(\bullet m(c)), \quad (1.3)$$

which is still satisfied by the original model, since $\bullet m(c)$ refers to Ed Koch even in the context of the belief operator.

We revise the substitution rule in the following way. Let ϕ_x^a stand for the substitution of a for the free variable x in ϕ .

$$(B\phi)_x^t = \begin{cases} B\phi_x^{\bullet t} & \text{if } t \text{ is not a bullet construction} \\ B\phi_x^t & \text{otherwise.} \end{cases} \quad (1.4)$$

C.2 Reduction theorems

A key notion for our development is that of a *reduction theorem* for a modal logic A . Basically, such a theorem shows how to reduce the unsatisfiability of a set of modal literals Z to the unsatisfiability of a set of sentences W whose modal depth is strictly less than that of Z . For example, consider the simplest case, the propositional belief logic K for a single agent. It is easy to prove that the set of modal atoms $Z = \{B\Gamma, \neg B\Delta\}$ is K -unsatisfiable if and only if for some $\delta \in \Delta$ the set $W = \{\Gamma, \neg\delta\}$ is K -unsatisfiable. Hence the unsatisfiability of Z is reducible to the unsatisfiability of W , and the modal depth of W is at least one less than that of Z .

For some logics, such as $S4$ and $S5$, it is not easy to find a reduction in terms of the modal depth of formulas, which is a syntactic property. Instead, we define a more semantic characterization of reduction in the next subsection.

C.3 Unsatisfiability depth for Kripke models

Consider a logic A and a sentence S . Suppose a model m satisfies S , so that $A \models_m S$. Now we may only have to search a certain part of m 's possible-world structure to establish the truth of S ; for example, in the epistemic logic $S4$, the truth of $S = Bp \supset BB\phi$ can be established for any m by traversing paths on the accessibility relation only to a depth of two. Paths longer than this have no role in determining the truth value of S .

To make this more precise, we introduce the concept of *agreement trees*. Let m be a model $\langle W, R, w_0, D, V \rangle$. A model m' is an agreement tree for m to depth n if the following conditions hold:

1. The structure of R' is a tree.
2. There is a one-one correspondence between paths of length less than or equal to n in the two models.
3. If $w_0 \dots w_{j-1} w_j$ is a path in m (with $j \leq n$), and $w'_0 \dots w'_{j-1} w'_j$ is its corresponding path in m' , then the domain and valuation of w_j and w'_j are the same, and $w_0 \dots w_{j-1}$ and $w'_0 \dots w'_{j-1}$ are also corresponding paths.

The agreement tree "unwinds" any cyclic structure of R to a depth of n . Note that the rest of the agreement tree can be arbitrary, i.e., it need not correspond to m .

Definition C.1 A set of sentences Γ is A -unsatisfiable at depth n if for every A -model m , every agreement tree of depth n falsifies some element of Γ .

We will write unsat_n to indicate unsatisfiability at depth n . Note that if a set is unsat_n , it is also unsat_k for all $k > n$, and also (simply) unsatisfiable.

C.4 Reduction theorems for epistemic logics

We now give reduction theorems for the six epistemic logics.

Definition C.2 The bullet transform of a set of formulas W is a set $W^\bullet$ derived from W by replacing all occurrences $\bullet t$ of the bullet construction with either $\bullet n(t)$ (if $\bullet t$ is under the scope of a modal operator) or $n(t)$ (if it is not), where n is function not occurring in W ; e.g., $\phi(\bullet a) \wedge B\phi(\bullet a) \rightarrow \phi(n(a)) \wedge B\phi(\bullet n(a))$. The identity transform of W is a set W^I formed by deleting all bullet constructors not under the scope of a modal operator, e.g., $\phi(\bullet a) \wedge B\phi(\bullet a) \rightarrow \phi(a) \wedge B\phi(\bullet a)$.

Theorem C.1 Let Z be a first-order satisfiable set of literals $\{\Sigma, B\Gamma, \neg B\Delta\}$ of an epistemic logic A , where Σ are nonmodal, and all bullet terms occurring in Δ also occur in Γ . Z is unsat_n if and only if for some $\delta \in \Delta$,

$$\left. \begin{array}{ll} (K) & \{\Gamma, \neg\delta\}^\bullet \\ (K4) & \{\Gamma, \neg\delta, B\Gamma\}^\bullet \\ (K45) & \{\Gamma, \neg\delta, B\Gamma, \neg B\Delta\}^\bullet \\ (T) & \{\Gamma, \neg\delta\}^\bullet \text{ or } \{\Gamma^I, \Sigma\} \\ (S4) & \{\Gamma, \neg\delta, B\Gamma\}^\bullet \text{ or } \{\Gamma^I, \Sigma\} \\ (S5) & \{\Gamma, \neg\delta, B\Gamma, \neg B\Delta, \neg B\neg\Sigma\}^\bullet \text{ or } \{\Gamma^I, \Sigma\} \end{array} \right\} \text{ is } \text{unsat}_{n-1}.$$

An example:

$$\begin{array}{ll} \{\neg B\neg B(p \wedge q), \neg Bp\} & \text{is } S5\text{-unsat}_2 \\ \{\neg B\neg B(p \wedge q), B(p \wedge q), \neg Bp\} & \text{is } S5\text{-unsat}_1 \\ \{\neg B\neg B(p \wedge q), B(p \wedge q), p \wedge q, \neg Bp, \neg p\} & \text{is } S5\text{-unsat}_0 \end{array}$$

D Analytic tableaux and completeness

We now give a brief overview of prenex analytic tableaux, which are defined in Smullyan [107]. Let S be a finite set of sentences in prenex form (all quantifiers precede other operators). A prenex tableau for S is a sequence of sentences starting with S , and containing instances derivable by the rules:

$$\frac{\forall x.\phi}{\phi_x^t} \quad \frac{\exists x.\phi}{\phi_x^t}, \text{ with proviso.}$$

In the existential rule, the proviso is that the term t has not yet been introduced in the tableau.

A prenex tableau is *closed* if some finite subset of its ground sentences is truth-functionally unsatisfiable. It is provable that the (perhaps infinite) set of sentences of an open prenex

tableau are first-order satisfiable. This yields a version of the Skolem-Herbrand-Gödel theorem for first-order logic: a set of sentences in prenex form is unsatisfiable if and only if a finite set of its instances is.

For a modal logic A , prenex form is the same as in first-order logic, taking modal formulae as unanalyzed predications. Thus $\forall x B \exists y Pxy$ is in prenex form; note that quantifiers which are under the scope of modal operators are *not* affected. We modify the definition of *closed prenex tableau* to be: some finite subset of its ground sentences is A -unsatisfiable. The key theorem for modal prenex tableaux is the following.

Theorem D.1 *If a finite set S of prenex sentences is A -unsatisfiable, then there exists a closed prenex tableau for S .*

We give a brief proof sketch of Theorem D.1. The proof is by induction on the unsatisfiability level of the prenex tableau. If S is unsat_0 , then by the results of Smullyan [107], its prenex tableau closes. Now assume that all sets that are unsat_{n-1} have closed prenex tableau. Let S be unsat_n . Suppose S has an open prenex tableau; consider the set W of all ground sentences on this branch. This set is first-order satisfiable. By elementary rules of propositional logic, and the reduction theorem for A , some set W' must be A - unsat_{n-1} . Now we convert this set to prenex form, and again set up a prenex tableau for W' ; this must close by the induction hypothesis for some finite $W'' \subseteq W'$; hence S cannot have an open tableau. Thus every set S which is A - unsat_n for finite n has a closed prenex tableau. Finally, if S is A -unsatisfiable, it is A - unsat_n for some n less than the maximum depth of embedding of modal operators; hence it must have a closed prenex tableau.

As an obvious corollary, we have the Skolem-Herbrand-Gödel theorem for A .

E B -resolution

Using the results of the previous section, we can now give a resolution method for the epistemic logics, which we call B -resolution.

F Clause form

Converting to clause form is the same as for first-order logic, with modal atoms having different argument structures treated as if they were different predicate symbols. Thus $B\forall x.P(x)$, BPa , and $B\exists x.P(x)$ are all considered to be different nilary predicates. Modal atoms with n free variables are n -ary predicates, *e.g.*, $B(P(x) \wedge \exists y.P(y))$ and $B(\exists y.P(y) \wedge P(x))$ are different unary predicates with the free variable x . Variables quantified under the scope of the modal operator remain unanalyzed or inert in B -resolution, and do not interact with variables quantified outside the operators. An example:

$$\forall x \exists y.P(x, y) \supset B \exists z.Q(x, y, z) \Rightarrow \neg P(x, f(x)) \vee B \exists z.Q(\bullet x, \bullet f(x), z)$$

Note that substitution of $f(x)$ for y in the modal context is done with $\bullet f(x)$. Also, in clause form we automatically insert a bullet operator before quantified-in variables (like x), to distinguish them from variables whose quantifiers are inside the scope of modal operators (like z).

We have proven the following theorem:

Theorem F.1 *A sentence is A-unsatisfiable if and only if its clause form is.*

G B-resolution

Our resolution method is based on Stickel's *total narrow theory resolution* rule [110], which has the following form. Let L be a language that embeds a theory T , that is, the axioms of T contain a set of predicates P of L (but not necessarily all predicates of L). Suppose there is a decision procedure for determining a set of ground literals W in P to be unsatisfiable (according to T). Then

$$\frac{\begin{array}{c} L_1 \vee A_1 \\ L_2 \vee A_2 \\ \vdots \\ L_n \vee A_n \end{array}}{A_1 \vee A_2 \vee \dots \vee A_n}, \text{ when } \{L_1, L_2, \dots, L_n\} \text{ is } T\text{-unsatisfiable} \quad (1.5)$$

is a resolution rule that is sound and complete for the theory T . This rule includes binary resolution as a special case, where L_1 and L_2 are complementary literals.

For epistemic logic A , the reduction theorem tells us when a set of literals will be A -unsatisfiable. Hence we can rephrase this rule is rephrased as follows. Let $\Gamma = \{\gamma_1, \gamma_2, \dots\}$ and $\Delta = \{\delta_1, \delta_2, \dots\}$ be finite sets of sentences, and $\Sigma = \{\sigma_1, \sigma_2, \dots\}$ a finite set of literals. In the case of ground clauses, we have the following two resolution rules:

$$\frac{\begin{array}{c} B\gamma_1 \vee A_1 \\ B\gamma_2 \vee A_2 \\ \vdots \\ \neg B\delta_1 \vee A'_1 \\ \neg B\delta_2 \vee A'_2 \\ \vdots \\ \sigma_1 \vee A''_1 \\ \sigma_2 \vee A''_2 \\ \vdots \end{array}}{A_1 \vee A_2 \vee \dots \vee A'_1 \vee A'_2 \vee \dots \vee A''_1 \vee A''_2 \vee \dots, \quad \text{where} \quad \left. \begin{array}{l} (K, T) \quad \{\Gamma, \neg\delta_1\}^\bullet \\ (K4, S4) \quad \{\Gamma, B\Gamma, \neg\delta_1\}^\bullet \\ (K45) \quad \{\Gamma, B\Gamma, \neg\delta_1, \neg B\Delta\}^\bullet \\ (S5) \quad \{\Gamma, B\Gamma, \neg\delta_1, \neg B\Delta, \neg B\neg\Sigma\}^\bullet \end{array} \right\} \text{ is unsat} \quad (1.6)$$

$$\frac{B\phi \vee A}{\phi^I \vee A} \quad (1.7)$$

In the first rule, we have listed all of the possibilities for the different epistemic logics. For the simplest case, K , only the clauses with Γ and δ_1 are used. The second rule is applied only for the knowledge logics T , $S4$, and $S5$.

H Lifting

The resolution rules have been given only for the ground case. Because of Theorem D.1, these rules will be complete if we are allowed to derive instances of any clause. Of course, this is a very inefficient way to do resolution, which is why unification is such an important concept. In this respect, B -resolution is more complicated than ordinary binary resolution, because there may be no "most general" unifier covering all possible ground resolutions. For example, consider the following two clauses:

$$\begin{array}{l} B(p(\bullet a) \wedge p(\bullet b)) \\ \neg Bp(\bullet x) \end{array} \quad (1.8)$$

There are two substitutions for x which yield a resolvent (a/x and b/x), but no most general unifier.

A second problem is that (1.6) and (1.7) are not true deduction rules, in the sense that they are not effective. The solution to this and the instantiation problems lies in how we check the unsatisfiability conditions. Suppose, each time we wish to do a B -resolution, we start another refutation procedure using the indicated sets of sentences. Then we intermix the execution of deductions in the main refutation proof with execution in the subsidiary ones being used to check unsatisfiability. If at some point a subsidiary refutation succeeds, we can construct a resolvent in the main refutation. If in addition we use a subsidiary refutation procedure that allows free variables in the input (essentially doing schematic refutations), then it is possible to subsume many instances of the application of the resolution rules in one unsatisfiability check. The details of this approach are discussed in Geissler and Konolige [27].

I Discussion

We are interested in general methods for finding resolution proof procedures for quantified modal logics. As this paper shows, one such method is to prove a reduction theorem for the logic. The nature of the reduction is apparent in the resolution rules, where unsatisfiability of a set of modal literals is reexpressed in terms of unsatisfiability of their arguments. We believe that such resolution methods are a natural and conceptually transparent means of finding refutations. A large part of the advantage comes from being able to strip off the modal operator and perform deductions on its arguments.

For the epistemic logics, reduction theorems are available. It is not clear that reduction theorems will always be provable for a modal logic. For example, if we add a common knowledge operator to an epistemic logic (see Halpern and Moses [38]), the resulting system is much more complicated, and it is an open question as to whether a reduction theorem exists.

Temporal logics are another important class of modal systems. Abadi and Manna [1] and Fariñas-del-Cerro [24] have both defined resolution systems for propositional temporal logics. However, their methods are not readily extendable to the quantified case. It would be interesting to try to use the techniques of this paper to formulate an alternative resolution system, and compare them.

Chapter 2

A RESOLUTION METHOD FOR QUANTIFIED MODAL LOGICS OF KNOWLEDGE AND BELIEF

This research was reported in the Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, California, 1986. It is the joint work of Kurt Konolige (SRI) and Christophe Geissler (l'École Nationale Supérieure des Telecommunications, Paris, France).

A Introduction

Modal logics with a possible-world semantics have been widely used to formalize various accounts of belief and knowledge in Artificial Intelligence (AI) (Moore [84], Levesque [67], and McCarthy [77]) and more recently in Computer Science in general (Halpern and Moses [38]). These logics are important both as an analytic tool in analyzing systems, and as a means of endowing artificial agents with the ability to reason about the knowledge and belief of other agents. In this latter category we include query answering (Levesque [67]), dialogue understanding (Appelt [6], Cohen and Perrault [13], and Grosz [37]), and multiagent planning systems (Konolige [50], Rosenschein and Genesereth [98]). It is widely recognized (see, for example, Moore [84]) that efficient and conceptually transparent proof methods are needed for these systems. By *efficient* we mean that computer automation of the methods produces those proofs needed for reasoning about belief within allowable time and space limitations; by *conceptually transparent* we mean that the action of the theorem prover is readily understandable, and the proofs clear and direct, so that it is easy to check and modify the behavior of the system.

While there has been a good deal of useful work on decision procedures for propositional modal logics (see Halpern and Moses [39]), fewer results have been obtained for quantified modal logics. Hilbert-style and natural deduction axiomatizations (Kuo [57]) exist, but there are no serious proposals to automate them. As an alternative, McCarthy [77] proved

theorems about a modal system by axiomatizing its possible-worlds semantics in a first-order system; subsequently Moore [84] used this technique to efficiently automate proofs. However, this is not a direct proof technique, because it involves reasoning about possible worlds and other objects of the semantic domain, rather than manipulating beliefs directly.

In this paper we present an efficient, direct proof method for a modal logic of belief that is based on Robinson's resolution principle ([96]). First we briefly review the modifications to first-order resolution that are necessary to establish the *B*-resolution rule. This rule has several properties which present problems for implementation in an automatic theorem-prover: it is non-effective, so we must find a way to apply it incrementally; and we must also develop techniques for controlling the size of the search space it generates.

The key idea we use to solve both problems is the concept of *semantic attachment* (Weyhrauch [115]). To illustrate this technique, consider the statements "*A* believes *P*" and "*A* doesn't believe $P \vee Q$." These are inconsistent in possible-worlds semantics, because there is no world compatible with *A*'s beliefs in which *P* is true and $P \vee Q$ is false. We can show this inconsistency by deducing a contradiction from *P* and $\neg(P \vee Q)$. Thus we can prove facts about belief statements by attaching to their meaning and performing a computation (in this case, deduction). The structure of reasoning is clear, and it is easy to understand and control the often confusing embedding of agents reasoning about other agents' beliefs.

B The Resolution Method

C Language preliminaries

We assume a modal language *L* built on a first-order language with function symbols. The modal atoms are of the form $[S]\phi$, where *S* is a term denoting an agent and ϕ is a formula denoting a proposition. The intending meaning is that *a* believes or knows ϕ .

The semantics of *L* are the standard Kripke possible-world models with an accessibility relation for each agent. It is well-known that various properties of knowledge and belief can be expressed by placing conditions on the accessibility relations (Halpern and Moses [39]). For simplicity of exposition we will limit ourselves to the system *K*, which has no restrictions, although versions of the resolution method have been derived for all the important systems (*T*, *K*4, *S*4, *K*5, *K*45, *S*5, etc.).

For technical reasons we make one further assumption: the domain of each possible world is a subset of the domain of any accessible world. Rescinding this restriction is possible, but introduces further complications in the resolution method that we do not wish to address here.

D Herbrand's Theorem

One version of Herbrand's Theorem is: *a set of universal sentences is unsatisfiable if and only if a finite subset of its instances are*. Stated in this form, it sanctions the "lifting" of proofs over ground sentences to those with universal variables. Unfortunately, Herbrand's Theorem is not true for modal logics with Kripke semantics, as we can see from the following counterexample:

$$\begin{aligned} &P(m(c)) \\ &\neg[S]P(m(c)) \\ &\forall x.Px \supset [S]Px \end{aligned} \tag{2.1}$$

We can construct a model as follows. Let P be the property of being non-Italian, let $m(x)$ denote the mayor of the city x , and c denote New York. Suppose S believes the mayor of New York is Fiorello LaGuardia (and not Ed Koch, the actual mayor); it is easy to confirm that all the sentences are satisfied.

Now if we substitute $m(c)$ for x in the third sentence, the resulting set is unsatisfiable. The reason is that, although x must refer to the same individual in all possible worlds, the substituted expression $m(c)$ need not. So even if a universal sentence is true in a model, some of its instances can be false.

Our solution to this problem is to redefine the meaning of "instance" by introducing a *bullet operator* ($\bullet$) whenever there is a substitution for variables inside the context of modal operators. $\bullet t$ always refers to whatever t denotes in the actual world, no matter what the context of interpretation; the bullet operator thus acts like a rigid designation operator for terms. In the above example, substituting $m(c)$ for x yields

$$P(m(c)) \supset [S]P(\bullet m(c)), \tag{2.2}$$

which is still satisfied by the original model, since $\bullet m(c)$ refers to Ed Koch even in the context of S 's beliefs.

With this revised definition of substitution (and instance), Herbrand's Theorem is once more valid (see Konolige [53]).

E Clause form

Converting to clause form is the same as for first-order logic, with modal atoms having different argument structures treated as if they were different predicate symbols. Thus $[S]\forall x.Px$, $[S]Pa$, and $[S]\exists x.Px$ are all considered to be different nilary predicates. Modal atoms with n free variables are n -ary predicates, e.g., $[S](Px \wedge \exists y.Py)$ and $[S](\exists y.Py \wedge Px)$ are different unary predicates with the free variable x . Note that variables quantified under the scope of the modal operator remain unanalyzed or inert in B -resolution, and do not interact with variables quantified outside the operators. An example:

$$\forall x \exists y Rxy \supset [S] \exists z. Rxyz \Rightarrow \neg R(x, f(x)) \vee [S] \exists z. R(\bullet x, \bullet f(x), z) \quad (2.3)$$

Note that substitution of $f(x)$ for y in the modal context is done with $\bullet f(x)$. Also, in clause form we automatically insert a bullet operator before quantified-in variables (like x), to distinguish them from variables whose quantifiers are inside the scope of modal operators (like z).

F B_K -resolution

Our resolution method is based on Stickel's *total narrow theory resolution* rule [110], which has the following form. Let L be a language that embeds a theory T , that is, the axioms of T contain a set of predicates P of L (but not necessarily all predicates of L). Suppose there is a decision procedure for determining a set of ground literals W in P to be unsatisfiable (according to T). Then

$$\begin{array}{c} L_1 \vee A_1 \\ L_2 \vee A_2 \\ \vdots \\ L_n \vee A_n \\ \hline A_1 \vee A_2 \vee \dots \vee A_n, \text{ when } \{L_1, L_2, \dots, L_n\} \text{ is } T\text{-unsatisfiable} \end{array} \quad (2.4)$$

is a resolution rule that is sound and complete for the theory T . This rule includes binary resolution as a special case, where L_1 and L_2 are complementary literals.

For the modal logic K , this rule is rephrased as follows. Let Γ be a set of formulas of L ; by Γ^* we mean the same formulas with the bullet operator uniformly replaced by a unary function not appearing in Γ . Then, in the case of ground clauses,

$$\begin{array}{c} [S]\phi_1 \vee A_1 \\ [S]\phi_2 \vee A_2 \\ \vdots \\ [S]\phi_n \vee A_n \\ \neg[S]\delta \vee A \\ \hline A_1 \vee A_2 \vee \dots \vee A_n \vee A, \text{ when } \{\phi_1, \phi_2, \dots, \phi_n, \neg\delta\}^* \text{ is } K\text{-unsatisfiable.} \end{array} \quad (2.5)$$

is a sound resolution rule for K . If we are allowed to infer instances of any clause, then by Herbrand's Theorem for L it is also a complete rule. Because it is a rule for the logic K , we call this the B_K -resolution rule.

G Implementation problems

The following problems must be solved to obtain an efficient implementation of B_K -resolution.

1. There is no decision procedure for unsatisfiability in quantified K .
2. Although we have given the resolution rule for the ground case, to be useful it must also be able to handle free variables in the arguments of the modal atoms. In this respect, B_K -resolution is more complicated than ordinary binary resolution, because in general there is no most general unifier covering all possible ground resolutions. For example, consider the following two clauses:

$$\begin{array}{l} [S](P \bullet a \wedge P \bullet b) \\ \neg[S]P \bullet x \end{array} \quad (2.6)$$

There are two substitutions for x which yield a resolvent (a/x and b/x), but no "most general" unifier.

3. The search space is exponential in the number of modal literals. Consider the following example:

$$\begin{array}{l} [S]r \vee A_1 \\ [S]p \vee A_2 \\ [S](p \supset q) \vee A_3 \\ \neg[S]q \\ \hline A_1 \vee A_2 \vee A_3 \end{array} \quad (2.7)$$

Only the last three clauses are needed for the resolution; indeed, including the first clause will not lead to a proof if A_1 cannot eventually be resolved away. In order to be complete in general theory resolution rules must be applied to a *minimal* set of unsatisfiable literals. If there are n clauses containing one modal literal each, there are 2^n possible B_K -resolutions that must be tried.

4. The above search space problem is compounded by the presence of variables, since a given clause may have to be used twice. For example, there is a resolution of the clauses

$$\begin{array}{l} Px \vee [S]P \bullet x \\ \neg[S](P \bullet a \wedge P \bullet b) \end{array} \quad (2.8)$$

yielding the resolvent $Pa \vee Pb$. However, this requires the first clause to be used twice in the belief resolution rule (2.5), as follows:

$$\begin{array}{l} Pa \vee [S]P \bullet a \\ Pb \vee [S]P \bullet b \\ \neg[S](P \bullet a \wedge P \bullet b) \\ \hline Pa \vee Pb \end{array}$$

5. If there are several clauses with negative belief literals for the same agent, we may duplicate our efforts in deciding unsatisfiability each time. Consider again example (2.7), and suppose there is another clause with the negative belief literal $\neg[S](q \wedge p)$. A resolution using this clause and the positive belief clauses exists; however, in finding it we duplicate the work involved in deciding that $\{p, p \supset q, q\}$ is unsatisfiable.

H A proof procedure for B_K -resolution

I Semantic attachment

We now give a version of B_K -resolution which treats the problems just mentioned. The key idea is to replace the unsatisfiability condition of (2.5) with a recursive call to the theorem-prover, using as input the arguments of the modal atoms. If the recursive call is successful, then the resolution rule can be applied. Because it is not certain that the call will terminate, processing of the call must be interspersed with other activities of the theorem-proving process. At any given time, the theorem prover must "time-share" its attention between ordinary binary resolution and multiple invocations of the semi-decision procedure.

In addition, we structure the semi-decision procedure so that it accepts free variables in formulas, and eventually returns substitutions covering all proofs that can be found with instantiations of these variables.

The idea of showing validity or unsatisfiability of a predication by means of a computation that reflects the intended meaning of the predicate is called *semantic attachment* (Weyhrauch [115]). In belief resolution, we compute the unsatisfiability of a set of modal literals by performing deductions on their arguments. This process is a generalization of semantic attachment in two ways. First, we show the unsatisfiability of a *set* of modal literals, rather than a single atom. Second, by allowing variables, we are able to perform many different instances of semantic attachment at once. Without this ability, belief resolution would not be efficient in the presence of variables, because we would have to first choose an instantiation of the modal literals without knowing whether it would lead to a resolution or not.

J An example

Our implementation of (2.5) has much in common with Kripke's [56] device of auxiliary tableaux. We define a structure called a *view*, which is an annotated instantiation of the theorem-proving process. Here is a short example to illustrate the basic idea. Assume initial clauses:

1. $[S]Pa$
2. $\neg Pb$
3. $Qx \vee Px \vee [S]P\bullet x$
4. $\neg[S](Pa \wedge P\bullet y) \vee Qy$
5. $\neg Qb$

Note that we have added a bullet operator to each variable under the scope of a belief atom. Ordinary resolution work as usual, for example, 2 and 3 can be resolved to yield:

6. $Qb \vee [S]P\bullet b$ 2,3

Clause 4 contains a negative belief literal, and we open a new view in an attempt to resolve it:

$$\frac{\text{view } S, \text{rems } (0, Qy)}{1. \neg Pa \vee \neg Pn(y) \vee \text{Ans}(0, y)}$$

This is view *for S*, the agent of the belief. The clause is derived from $\neg(Pa \wedge P \bullet y)$; note the substitution of the function *n* for the bullet operator. The *Ans* predicate keeps track of the input free variable *y*; it also contains the additional argument "0" to indicate that it is connected to the remainder (*rems*) indexed by 0. If a proof is found in the view, the remainder *Qy* will be returned with an appropriate binding for *y* as a deduced clause of the original proof.

We can add the arguments of positive belief atoms to the view, as in clause 1 (of the original clause set). The view now contains:

$$\frac{\text{view } S, \text{rems } (0, Qy)}{1. \neg Pa \vee \neg Pn(y) \vee \text{Ans}(0, y) \\ 2. Pa}$$

These two clauses can be resolved, yielding:

$$\frac{\text{view } S, \text{rems } (0, Qy)}{1. \neg Pa \vee \neg Pn(y) \vee \text{Ans}(0, y) \\ 2. Pa \\ 3. \neg Pn(y) \vee \text{Ans}(0, y) \quad 1, 2}$$

Clause 6 has a positive belief literal, so we add its argument also:

$$\frac{\text{view } S, \text{rems } (0, Qy) (1, Qb)}{1. \neg Pa \vee \neg Pn(y) \vee \text{Ans}(0, y) \\ 2. Pa \\ 3. \neg Pn(y) \vee \text{Ans}(0, y) \quad 1, 2 \\ 4. Pn(b) \vee \text{Ans}(1)}$$

Clause 4 contains an answer predicate with a new index. The remainder of the original clause containing the positive belief atom (6) is inserted into the indexed remainder list. Note that the bullet operator was replaced with the same function *n* as in clause 1.

Clauses 3 and 4 resolve, yielding a clause containing just answer predicates:

$$\frac{\text{view } S, \text{rems } (0, Qy) (1, Qb)}{1. \neg Pa \vee \neg Pn(y) \vee \text{Ans}(0, y) \\ 2. Pa \\ 3. \neg Pn(y) \vee \text{Ans}(0, y) \quad 1, 2 \\ 4. Pn(b) \vee \text{Ans}(1) \\ 5. \text{Ans}(0, b) \vee \text{Ans}(1) \quad 3, 4}$$

Now we gather up the remainders indexed by the answer predicates in the answer clause, namely, Qb (index 1) and Qy (index 0). Using the substitution b/y generated by the *Ans*-predicate, we return $Qb \vee Qb (= Qb)$ as the result.

1. $[S]Pa$
2. $\neg Pb$
3. $Qx \vee Px \vee [S]P \bullet x$
4. $\neg[S](Pa \wedge P \bullet y) \vee Qy$
5. $\neg Qb$
6. $Qb \vee [S]P \bullet b$ 2, 3
7. Qb 1, 4, 6

Clauses 5 and 7 resolve to give the null clause, completing the proof.

K Views

Formally, a view is an annotated, finite set of clauses. The annotation is a list of remainders to be used in returning a result from the view. We perform four operations on views.

Opening a view. Let C be a clause of the form $\neg[S]\phi \vee A$. A view for S may be created. Into it we insert clauses formed from ϕ as follows. Let W be the set of clauses resulting from putting $(\neg\phi)^*$ into clause form, and let x be the free variables of ϕ . We insert each member of W into the view, disjoining the answer predicate $Ans(0, x)$. We also add the annotation $(0, A)$ to the remainder list.

Adding a positive belief literal. Let C be a clause of the form $[S]\phi \vee A$, and let x be the free variables of ϕ . To any existing view for S we may add the clauses formed by converting ϕ^* to clause form and disjoining $Ans(n, x)$, where n is a new index. (n, A) is added to the remainder list.

Stepping a view. A resolution step may be performed in any view. This includes using one of the four operations described here to create and manipulate embedded views.

Returning an answer. If a clause containing only answer literals is deduced in a view, we may assert a new clause in the proof containing the view. Let

$$\begin{aligned}
 & Ans(0, a) \vee \\
 & Ans(n_1, a_1^1) \vee \cdots \vee Ans(n_1, a_{i_1}^1) \vee \\
 & \vdots \\
 & Ans(n_k, a_1^k) \vee \cdots \vee Ans(n_k, a_{i_k}^k)
 \end{aligned}$$

be the answer clause, and let $A_n(a)$ be n^{th} remainder with a substituted for its free variables (a may itself contain variables). The returned clause is:

$$\begin{array}{l}
A_0(\mathbf{a}) \vee \\
A_{n_1}(\mathbf{a}_1^1) \vee \quad \dots \quad \vee A_{n_1}(\mathbf{a}_{i_1}^1) \\
\vdots \\
A_{n_k}(\mathbf{a}_1^k) \vee \quad \dots \quad \vee A_{n_k}(\mathbf{a}_{i_k}^k)
\end{array}$$

Note that only one $Ans(0, \mathbf{a})$ -predicate is allowed in the answer clause. Multiple answer predicates are allowed for positive belief atoms, because more than one instance of these atoms may participate in B_K -resolution.

The use of the Ans -predicate allows us to perform a schematic proof, where the input sentences can have free variables. At the end of a proof, the answer predicates give the necessary instantiations of the free variables. Thus we have "lifted" B_K -resolution from the ground case.

If these rules are added to a refutation system using ordinary resolution, we can prove the following result. Let W be a set of clauses of L , and suppose there is a set of ground instances $W\theta$ such that B_K -resolution derives the ground clause C . Then there is a sequence of applications of the view rules on W that returns a clause C' having a ground instance C . Thus these rules faithfully implement B_K -resolution, and together with ordinary resolution form a sound and complete system for K .

L Agent terms

We have implicitly assumed that in modal atoms of the form $[S]\phi$, S is a ground term. However, we may easily lift to the more general case of variables, because the agent term is not in a modal context. There are two modifications to the rules. First, in opening a view, $\mathbf{x}$ is a list of all variables in both ϕ and S . Second, we may add a positive belief literal $[S']\psi$ to any view for S , if S and S' have a most general unifier θ . When adding clauses obtained from ψ , we must also disjoin the answer predicate $Ans(0, \mathbf{x}\theta)$. This is to assure that a result is returned only if all the participating clauses have unifiable agent terms.

M Controlling the search space

N Avoiding redundancies

We now address the implementation problems mentioned in the previous section. All of the methods mentioned here maintain the soundness and completeness of B_K -resolution.

1. The view rules split each possible B_K -resolution into a sequence of effective steps. These steps may be interspersed with other activities of the theorem-prover, including ordinary resolution.

2. The use of answer predicates allows a schematic proof within views, so that free variables in the input can be tolerated. Separate proofs are found whenever there is no unifying instance of the input variables that allows a single schematic proof. Consider again example (2.6). If we open a view for the negative belief atom, and add the positive one, we get:

$$\begin{array}{l} \text{view } S, \text{ rems } (0, x) \\ 1. \neg Pn(x) \vee Ans(0, x) \\ 2. Pn(a) \\ 3. Pn(b) \end{array}$$

There are two proofs, one with a/x and one with b/x . Note that if there are no free variables or remainders when we add a clause, we can forgo the answer predicate.

3. We do not need to separately consider all possible combinations of modal literals that could lead to B_K -resolvents. The proof structure of the view takes care of this: only the remainders of those clauses that participated in the proof are returned in the result. Consider again example (2.7). We open a view for the negative belief literal, and add the arguments of all three positive belief literals. The view looks like this:

$$\begin{array}{l} \text{view } S, \text{ rems } (1, A_1) (2, A_2) (3, A_3) \\ 1. \neg q \\ 2. r \vee Ans(1) \\ 3. p \vee Ans(2) \\ 4. \neg p \vee q \vee Ans(3) \end{array}$$

Two resolutions yield $Ans(2) \vee Ans(3)$, returning the result $A_2 \vee A_3$. Although the clause $[S]r \vee A_1$ was added to the view, it was never used in the proof.

4. Although several instances of the same clause may be needed to form a B_K -resolvent, we need only add its belief literal *once* to the view. Consider again example (2.8). We open a view for the negative belief literal, and add the positive one, obtaining:

$$\begin{array}{l} \text{view } S, \text{ rems } (1, Px) \\ 1. \neg Pn(a) \vee \neg Pn(b) \\ 2. Pn(x) \vee Ans(1, x) \end{array}$$

By two resolutions of the second clause, we get:

$$\begin{array}{l} \text{view } S, \text{ rems } (1, Px) \\ 3. \neg Pn(b) \vee Ans(1, a) \quad 1, 2 \\ 4. Ans(1, b) \vee Ans(1, a) \quad 2, 3 \end{array}$$

This is a particularly nice result, since the necessity of using multiple copies of a clause in resolution gives rise to nasty control problems.

5. With a little care in indexing the *Ans*-predicates, we can eliminate the redundancies caused by performing the same deductions on the arguments of positive belief literals in different views. Suppose we create only one view for each agent S , but we allow any negative belief literal $\neg[S]\phi$ to be added to this view, in the same way as positive literals are added. We keep track of the answer index so that that we can identify it as arising from a negative belief literal. Resolution are performed as usual within the view. However, to return an answer, we apply the following condition: exactly one *Ans*-predicate arising from a negative belief literal must appear in the answer clause. For example, consider the following clause set:

1. $[S]\forall x.Px$
2. $[S](\forall x.Px \supset Qx)$
3. $A_0 \vee \neg[S]Qa$
4. $A_1 \vee \neg[S]Qb$

We open a single view, inserting all belief literals:

- view S rems $(0, A_0) (1, A_1)$
1. Px
 2. $\neg Px \vee Qx$
 3. $\neg Qa \vee Ans(0)$
 4. $\neg Qb \vee Ans(1)$

Resolving 1 and 2 yields Qx , which can be resolved separately against 3 and 4, returning A_0 and A_1 , respectively. However, any resolutions which contain *both* 3 and 4 as ancestors will have $Ans(0)$ and $Ans(1)$ predicates, and so will not generate any result clauses.

The interesting point to note here is that we need open only a single view for each agent, instead of each negative belief literal. The view acts as a deductive testbed in which we try to show different combinations of belief and nonbelief are inconsistent for the agent.

It is possible to generalize this strategy to different agents sharing a set of common beliefs: a single view is created for all the agents. This is particularly useful when one has to deal with agent terms having variables, as in the following clause set:

1. $\neg Px \vee [x]q_1$
2. $\neg Px \vee [x](q_1 \supset q_2)$
- $\vdots$
- n . $\neg Px \vee [x](q_{n-1} \supset q_n)$
- $n + 1$. Pa
- $n + 2$. Pb
- $n + 3$. $\neg[a]q_n \vee \neg[b]q_n$

It is clear that a and b share many of the same beliefs, and that a great deal of effort will be saved if we assert these beliefs in the same view.

O Heuristic control

We have investigated several refinements of the view rules that do not maintain completeness, but may be useful heuristic methods for controlling the size of the search space.

The first is to limit the depth of recursion of views. In a particular problem domain we can often judge whether or not it is useful to reason about agents reasoning about agents reasoning about agents ... and so on. By refusing to open views that are embedded beyond a certain depth, we can control inferences about nested reasoning. More fine-grained control is also possible, if we know that certain types of nested reasoning will be more useful than others. For example, if introspective reasoning is not required (an agent reasoning about his or her own beliefs) then we can refuse to open a view for S if it is embedded in a view for S .

A second method of control is to integrate the view rules into a set-of-support strategy. The most obvious method is to open a view only for negative belief literals in the set of support. The rationale is that we often have a large number of facts about an agent's beliefs, and we are trying to prove from these that the agent has some other belief. A negative literal $\neg[S]\phi$ will appear in the set of support when we are trying to prove that S has the belief ϕ .

Unlike in ordinary resolution, this set-of-support strategy is not complete because it does not permit inferences about lack of belief. For example, we cannot infer $\neg[S]p$ from $[S](p \supset q)$ and $\neg[S]q$, because there are no negative belief literals in the set of support.

P Implementation

The view rules for quantified modal K have been implemented using a nonclausal connection-graph theorem prover developed by Stickel [109]. The implementation itself is of interest, especially the method of sharing the attention of the theorem-proving process between views (see Geissler and Konolige [27]).

In addition, we have incorporated theories of common belief, and a simple modal form of the situation calculus (McCarthy and Hayes [78]) as a logic of time. We have derived an automatic proof of the Wise Man puzzle that illustrates these ideas, showing the interaction between belief, action, and time. The proof is conceptually simple and easy to follow.

Q Other resolution systems for modal logics

Currently there are at least two other approaches to using resolution in a quantified modal logic, both for temporal logics. Fariñas-del-Cerro [24] describes a resolution method for restricted languages in which there are no quantifiers in modal contexts. Such languages are not suitable for knowledge and belief, because it is impossible to express, for example, the statement "Ralph knows that someone is a spy."

Abadi and Manna [1] derive sound and complete nonclausal resolution rules for propositional temporal logics, and are working on extending their techniques to the quantified case.

This work is interesting because it incorporates induction rules, a necessity for completeness in temporal logics containing both *next state* and *always* operators. When belief logics are extended to contain common belief operators, a similar problem surfaces (see Halpern and Moses [38]). We may be able to adopt a solution analogous to those found for temporal logics; currently we have only incomplete resolution rules for common belief.

A major difference between temporal logic resolution and the methods presented here is the use of semantic attachment. The temporal resolution rules are binary rules that transfer arguments in and out of the scope of modal operators; eventually a form results that can be resolved away. This type of resolution does not seem to result in perspicuous, easily-controlled proof methods.

Q.1 Acknowledgements

We are grateful to Mark Stickel for his help in modifying the connection-graph theorem prover.

Chapter 3

REPRESENTING DEFAULTS WITH EPISTEMIC CONCEPTS

This section was written by Kurt Konolige and Karen Myers. It will appear as a Stanford and SRI Technical Note.

A Introduction

Reasoning about defaults — implications that typically hold, but which may have exceptions — is an important part of commonsense reasoning. In this paper we address the problem of arriving at a theory of defaults that is in accord with our general intuitions, and expressing it in formal terms.

Where do intuitions about defaults come from? Certain aspects of defaults, for example, their defeasibility, are obvious from the way in which we talk about them. Other aspects of defaults relate to interactions among competing defaults; the presence of hierarchies introduces certain subtle constraints in adjudicating among these. In the first part of the paper, we will present various parts of a theory of defaults that we believe pose significant representational problems.

There have been several formalizations of some theory of defaults in the AI literature. Reiter [94] proposed a formal system called *default logic*, which uses a form of inference rule that refers to consistency in the resulting theory. McDermott and Doyle [79] use a modal operator for the same purpose, and call their system *nonmonotonic logic*. Autoepistemic logic [83] is a clarification (from a semantical point of view) and reconstruction of nonmonotonic logic using explicit epistemic concepts. Levesque [67] also uses an autoepistemic logic called KFOPC for reasoning about defaults. Finally, defaults have been formalized using the circumscription schema of McCarthy [75], which involves only first-order constructs.

All of these formalisms exhibit the property of inferential nonmonotonicity: a formula p which follows from a set of formulas Γ may not follow from a strictly larger set Γ' (indeed, its negation $\neg p$ may be a consequence). This fact is exploited to represent of some

aspects of defeasibility. However, other desiderata for a theory of defaults are not automatic consequences of these formalisms.

We concentrate on expressing our theory of defaults within autoepistemic logic, by providing a translation from the statement "typically A's are B's" into sentences of the logic. As we will show, this translation is faithful to our intuitions about the several different ways in which defaults can be defeated, as well as the constraints introduced by hierarchies.

There is a close connection between our derived constraints on defaults in the presence of hierarchies and the work of Touretzky [113] on inheritance in networks. We argue that Touretzky's *inferential distance* algorithm is a strengthening of our condition of hierarchic defeat, and as such, is unsound with respect to our theory.

B Defaults

By *default* we mean an assumption about the world that is made on the basis of current (incomplete) beliefs, and which may be contradicted by acquired information. There are many types and uses of defaults in AI; for example, the so-called *closed-world assumption* is one type of default, in which it is assumed that a database contains all of the true positive instances of relations. Here we are concerned with commonsense reasoning, expressed by statements of the form "typically A's are B's" (we abbreviate this as $A \rightarrow B$). As Reiter and Criscuolo [95] have pointed out, there are usually two connotations to the word "typical." One is probabilistic: saying "typically birds fly" entails that most birds fly. There is also a prototypical notion, in the sense that, having no contradictory information, we are willing to assume that any given bird is an instance of the prototypical bird that flies. This notion of prototype is part of the underlying intuition behind the concept of frames [82], in which individuals have properties by virtue of being prototypical instances of some class.¹

The idea that we can assume any given bird to fly, barring evidence to the contrary, is basically an *epistemic* one. To use a term from the philosophical literature, if we believe Tweety is a bird, we are *prima facie* justified in believing Tweety flies [93]. This justification is *defeasible*: an explicit belief that contradicts the conclusion will nullify the justification. Any proposition that, if believed, will nullify a default, is called a *defeater* of the default.

There are several subtle aspects of defeasibility and defeaters that are important for any representation of defaults; we discuss these in the next subsection. What defeaters actually exist for a given default relative to a domain of application is an important empirical study. It is useful to try to find general principles that hold across many different domains. We will motivate a very important constraint on defaults in the domain of property inheritance in hierarchies, which is of some interest to AI.

¹The notions of "most" and prototype need have no necessary connection. A prototype is what one is willing to assume in the absence of additional information; obviously, considerations of action or purpose will enter here. For instance, one would be wise to assume that the prototypical mushroom is poisonous, even though most mushrooms are not. Langlotz [60] has recently attempted to relate the concept of prototype to that of utility in decision theory. Most of the arguments raised in this paper require only that "typical" refer to prototypes, so they are applicable even when a minority of A's are B's.

B.1 Three types of defeat

Consider the reasoning that might be involved in deciding whether a car will start. One possible default is

If the car started yesterday, then it will start today. (3.1)

This is only a *prima facie* rule because there could be problems with the car that arose overnight — someone could have stuffed a potato in the tailpipe. It is known that whenever there is a potato in the tailpipe the car will not start. So believing that there is a potato in the tailpipe is a defeater for the default, because it contradicts its conclusion. Pollock [93] has called this Type I defeat.

Another type of defeat occurs if the default as a whole is undermined, rather than its conclusion. For example, suppose that the car started yesterday, but only because it was jump-started. We can now no longer conclude that the car will start today, even though the car started yesterday. The conclusion of the default has not been contradicted, because the car may indeed start; but the default is no longer applicable. This kind of defeat is called Type II by Pollock.

An interesting difference between these two defeat types occurs in the presence of multiple defaults supporting the same conclusion. Suppose the car has a dashboard indicator showing the state of its electrical system. Then another default might be

If the electrical system is ok, the car will start. (3.2)

Obviously, this default will not be defeated even if the car was jump-started. A Type I defeater, on the other hand, is a defeater for every *prima facie* rule whose conclusion it contradicts.

It is possible to distinguish a third type of default, which combines the characteristics of the first two. Suppose the car develops an intermittent electrical fault, which does not register on the dashboard indicator. The fault is a random process, and it is impossible to know beforehand whether the car will start on any particular occasion.² Such a fault defeats both defaults (3.1) and (3.2) above; indeed, it defeats *any* default whose conclusion is that the car starts. It thus shares this property with Type I defeaters. But it also has a property of Type II defeaters, in the sense that it does not directly contradict the conclusion; the car may indeed start. We call this type of defeat Type III. It presents the hardest representational difficulties, since it requires defeating all defaults of a certain type, without contradicting their conclusions.

B.2 Competing defaults and hierarchies

It often happens that defaults will compete or conflict with one another. A standard example is the two defaults:

²One author has actually had experience with such a car.

Typically Quakers are pacifists.
Typically Republicans are not pacifists. (3.3)

Nixon, a known Quaker and Republican, could be judged *prima facie* to be either a pacifist or not; if the conclusion of one default is accepted, it is a Type I defeater of the other. Such competing defaults must usually be adjudicated on the basis of domain evidence: whether the typical Republican Quaker a pacifist or not is a matter of empirical discovery. However, there are cases in which competing defaults can be resolved on the basis of hierarchic information.

Suppose A is a subset of B . Knowing that an individual a is in A is more informative than knowing she is in B , since the latter is entailed by the former. Hence any default whose antecedent is satisfied by members of A should prevail over a conflicting default satisfied by B . Examples of this sort abound — a standard one in the literature is drawn from biological taxonomy [22]. Molluscs typically have shells; cephalopods, a type of mollusc, typically do not; but nautili, a subclass of cephalopods, typically do. Obviously, being a cephalopod is a defeater (Type II) for the default that molluscs have shells, and similarly being a nautilus defeats the cephalopod default. This example is an instance of a general principle, which we phrase as follows. Let $A \subset B$ mean that A is a proper subset of B . In first-order logic, A and B are unary predicates, and $A \subset B$ abbreviates $\forall x [(A(x) \supset B(x)) \wedge \neg (B(x) \supset A(x))]$.

Principle of hierarchic defeat. If there are two defaults $A \rightarrow C_1$ and $B \rightarrow C_2$, such that (1) $A \subset B$ and (2) C_1 and C_2 are inconsistent, then A is a Type II defeater of the default $B \rightarrow C_2$.

C_1 and C_2 are inconsistent if the negation of one is the first-order consequence of the other.

The plausibility of this principle can be argued on probabilistic as well as intuitive grounds. Let us assume that the "typical" entails a certain probability threshold, so that $A \rightarrow C_1$ implies that more than $t\%$ of the A 's have the property C_1 . Now consider the set composed of both A 's and B 's; in order for it to be a default that a member of this set has property C_2 , it must be that at least $t\%$ of the individuals in this set have the property. This cannot be the case, since the set $A \cup B$ is just A , and $t\%$ of the A 's have property C_1 , which excludes them from having property C_2 .

It is interesting to note that a strengthening of the principle, in which the premise $A \subset B$ is weakened to $A \rightarrow B$ (not *all* A 's need be B 's), cannot be justified on similar probabilistic grounds. In this case, even though most A 's have property C_1 , a majority of those A 's which are also B 's may have property C_2 . Touretzky's inferential distance algorithm [113] for deciding among competing defaults, which entails this stronger principle, is thus incorrect in certain cases. For example, consider the following set of defaults:

$A \rightarrow C$
 $A \rightarrow A \wedge D$
 $A \wedge D \rightarrow \neg C$ (3.4)

Now by default an individual a who is A will also be $A \wedge D$, which is a subset of A . By the hierarchic defeat principle, we know that the default $A \wedge D$ defeats the default $A \rightarrow C$ for a . But by the inferential distance algorithm, the $A \rightarrow C$ will be applied, so that a will both $A \wedge D$ and C . The reason inferential distance is correct in the examples given in [113] is that the default $A \rightarrow B$ is always chosen so that A is a subset, or nearly so, of B . However, there is no necessary subset relation between A and B given the default; in this counterexample we have chosen a case where B is a subset of A .

B.3 Correct inference

The problem of correct inference in a default theory is this: given an initial belief set of "hard facts" and defaults, what additional inferences should an ideal agent make? If there are no defaults, the agent should infer all the logical consequences of the hard facts. But the notion of *correct* inference is complicated by the presence of defaults, because the inferences that are sanctioned by *prima facie* rules are only plausible, and can be defeated. Hence correct inference can only be defined by reference to the total set of inferences that it is possible to make. In addition to inferring the logical consequences of her beliefs, an agent, at a minimum, should infer the conclusion of any default whose premises are beliefs, and which is not defeated by any possible inference the agent could make. This comes from the very nature of defaults as *prima facie* rules, which must apply if there is no information which defeats them.

When there are several conflicting defaults that could be applied, as in the case of the Nixon example above, two possible choices for inference are to include a consistent set of default conclusions (*brave* inference in the terminology of [21]), or to exclude them entirely (*cautious* inference).

C Representation

In this section we take up the issue of representing our default theory in a formal system. Generally, this involves showing how to translate default statements like $A \rightarrow B$ into sentences of the formal system. There are two criteria on which to judge the appropriateness of the formalization:

1. The translation should be *local*.
2. The formal system should produce all and only the correct inferences.

The first criterion concerns the way in which statements of the form $A \rightarrow B$ are represented in the formal system. Suppose we are translating a set of defaults and "hard facts" V into sentences of the formal system. A *local* translation (see [54]) is one in which each default statement can be effectively translated into one or more sentences without regard to any other statements in V . Much of the difficulty in representing defaults is finding a local translation that preserves the defeasibility characteristics of defaults in the presence of other information.

C.1 Previous formalizations

As noted in the introduction, a number of formalizations exist for some notion of “default.” McCarthy [75] proposes the introduction of a predicate *ab* for “abnormality,” and represents $A \multimap B$

$$\forall x. A(x) \wedge \neg ab(x) \supset B(x). \quad (3.5)$$

Inference within the formal system is defined as the sentences true in all models minimal with respect to the *ab* predicate.

This formalization handles both Type I and II defeat quite nicely. For the latter, one asserts that an individual is abnormal, and the default is blocked without asserting anything that contradicts its conclusion.

There are some problems with this approach; Etherington [21] notes that in the absence of any information about particular birds, it implies that all *A*’s are *B*’s, which is clearly not a correct inference from the original default. From our point of view, a more serious shortcoming is that representing Type III defeat is problematic — there is no way to defeat *all* defaults whose conclusion is that an individual is *B*, without saying the individual is not *B*, or enumerating all of the associated abnormality predicates. This is because Type III defeat is really expressed by epistemic concepts, which are absent from this formal system.

Other approaches to defaults (default logic, nonmonotonic logic, KFOPC) have concentrated on representing only a single type of defeat, Type I. Although it is clear that the other types of defeat can be represented (for example, Reiter and Crisculo [95] show how to express Type II defeat within default logic), there has been little concern to provide a *local* translation of statements about defaults, or to find and represent general principles like hierarchic defeat. An exception to this is Levesque’s work in KFOPC [67], in which there is a deliberate attempt to preserve locality in the translation. Our idea for formalization is similar in spirit to his: we introduce an *ab* predicate into a logic which explicitly represents epistemic concepts.

C.2 Overview of AE logic

AE logic is a formalism for modeling an agent with the power to reason about its own beliefs. The language is first-order, augmented by the modal operator *L*. The intended meaning of a formula $L\phi$ is that ϕ is a belief of the agent. In this logic, an agent with an initial set of beliefs *A* would ideally believe a set of sentences *T* referred to as the *AE expansion* of *A* (see [83] for a comprehensive presentation). In addition to being logically closed, an AE expansion *T* has the following properties:

$$\begin{aligned} [(AE1)] \quad & P \in T \text{ iff } LP \in T \\ [(AE2)] \quad & P \notin T \text{ iff } \neg LP \in T \end{aligned} \quad (3.6)$$

The proof-theoretic characterization of AE expansions is given as a fixed-point definition. An expansion T is the set of all logical consequences of a base set A and the set of assumptions $\neg L\alpha$, where $\alpha \notin T$. Informally, one can consider arriving at T by making oracular assumptions about what is *not* believed, and adjoining them to A .

The potential for multiple expansions of an initial set of beliefs exists due to the fixed-point nature of AE expansions. Consider for instance the case where the base set $A = \{\neg LP \supset Q, \neg LQ \supset P\}$. Since nothing is known about P , we can postulate an expansion that does not contain P . By (AE2) then, it follows that that $\neg LP$ is in the belief set, and hence so is Q . Analogously we can apply the same argument to Q , producing an expansion that contains (among other things) $\neg LQ$ and P . This capability will introduce some difficulties in the formalization of defeat.

C.3 Formalizing Types I, II and III defeat

Consider now the canonical default: "Typically α 's are γ 's". This statement can be represented by the following implication schema:

$$\mathcal{D}. \quad L\alpha(x) \wedge \neg L\phi(x) \supset \gamma(x)$$

We will refer to $\mathcal{D}$ as the *default rule* for the given intuitive default. Here $\alpha(x)$ and $\gamma(x)$ are non-modal formulas applied to some named individual.

The introduced proposition ϕ serves as the *defeater* for the default in that the applicability of $\mathcal{D}$ depends on a lack of belief in ϕ . Any derivation of ϕ in an expansion would abrogate the default conclusion of $\gamma(x)$ from $\alpha(x)$. The three types of defeat are distinguished in their representations by the manner in which they produce the derivation of the defeater ϕ .

A Type I defeat, by definition, results when the negation of the conclusion of a default is derivable. The following implication schema thus captures the notion of this type of defeat:

$$\mathcal{DI}. \quad \neg\gamma(x) \supset \phi(x)$$

Type II defeat corresponds to the over-riding of a default as a whole, independent of the status of the conclusion. Typically, some sentence $\beta(x)$ is asserted to cut the support of $\alpha(x)$ for $\gamma(x)$. This is represented simply as:

$$\mathcal{DI}'. \quad \beta(x) \supset \phi(x)$$

With defeats of Type III, a default is blocked as a result of disbelief in the truth of its conclusion. Hence Type III defeat is characterized by:

$$\mathcal{DIII}. \quad \neg L\gamma(x) \supset \phi(x)$$

Note how this translation process for defaults and defeats satisfies the locality property advocated earlier.

Another point of significance is the fact that $\mathcal{DI}$ and $\mathcal{DIII}$ are integral parts of the representation of the default. Without their inclusion, certain initial belief sets have no expansions. For example, the base sets $A = \{\mathcal{D}[a], \neg L\gamma(a), \alpha(a)\}$ and $A = \{\mathcal{D}[a], \neg\gamma(a), \alpha(a)\}$ have no AE expansions ($\mathcal{D}[a]$ is the instantiation of schema $\mathcal{D}$ with x replaced by a).

C.4 The car example

As an illustration of these concepts, consider the set of defaults and assertions about cars introduced in the previous section:

- (1) If the car started yesterday, then typically it will start today.
- (2) If the electrical system is ok, the car will typically start.
- (3) If there is a potato in the tailpipe, the car will not start.
- (4) If the car was started yesterday by jump-starting it, then the fact that it started yesterday has no bearing on whether or not it will start today.
- (5) If the car has an intermittent electrical fault, then there is no way of knowing whether or not it will start today.

In the list above, (1) and (2) are defaults while the remainder are strict implications. This set can be formalized as:

$$\begin{array}{lll}
 \mathcal{D}_{(1)} & LStart\text{-}yesterday(x) \wedge \neg L\phi_1(x) & \supset Start(x) \\
 \mathcal{D}_{(2)} & LOK\text{-}electrically(x) \wedge \neg L\phi_2(x) & \supset Start(x) \\
 \mathcal{A}_{(3)} & Potato\text{-}in\text{-}tailpipe(x) & \supset \neg Start(x) \\
 \mathcal{A}_{(4)} & Jump\text{-}started\text{-}yesterday(x) & \supset \phi_1(x) \\
 \mathcal{A}_{(5)} & Intermittent\text{-}fault(x) & \supset \neg LStart(x) \wedge \neg L\neg Start(x) \\
 (*) & \neg LStart(x) \vee \neg Start(x) & \supset \phi_1(x) \wedge \phi_2(x)
 \end{array}$$

The formulas $\mathcal{D}_{(1)}$ and $\mathcal{D}_{(2)}$ are the default rules corresponding to the defaults (1) and (2) while the formulas $\mathcal{A}_{(3)}$ – $\mathcal{A}_{(5)}$ are translations of the remaining assertions. The formula (*) combines the instantiations of the Types I and III defeat schemas $\mathcal{DI}$ and $\mathcal{DIII}$ for $\mathcal{D}_{(1)}$ and $\mathcal{D}_{(2)}$. $\mathcal{A}_4$ is an instance of Type II defeat of $\mathcal{D}_{(1)}$.

C.5 The multiple expansions problem

This approach to representing defaults has the side-effect of licencing undesirable expansions. As an illustration, consider the case where $A = \{\mathcal{D}[a], \mathcal{DIII}[a], \alpha(a)\}$ for some individual a . There are two expansions for this A . In the first, the defeater $\phi(a)$ does not hold.

thus $\neg L\phi(a) \in T$ and $\gamma(a)$ is concluded. There is a second expansion in which $\neg L\gamma(a)$ is assumed to hold, thus resulting in a Type III defeat of the default. This latter expansion is counter-intuitive — in a sense we are defeating the default by assuming that its conclusion is not in the belief set. Clearly this violates one of the principles of correct inference for defaults, namely to allow the default conclusion unless there is explicit evidence contradicting it. The problem arises in expansions where the lack of belief in the conclusion ($\neg L\gamma(a)$) takes precedence over a lack of belief in the defeater ($\neg L\phi(a)$).

From another perspective, defeaters such as $\phi(a)$ (and hence defeat itself) should be *earned* rather than derived from arbitrary assumptions of disbelief. The unwarranted assumption of $\neg L\gamma(a)$ directly produces a Type III defeat of this second default via the instantiated *DIII* schema. Clearly one should reject expansions of this nature; we will refer to such expansions as *unsanctioned*.

AE logic provides no mechanism by which the preference of assumptions delineated above can be enforced. As a result, we appeal to extra-logical criteria in order to eliminate unsanctioned expansions.

Define a *disbelief atom* to be any formula preceded by the $\neg L$ operator, and let $>$ be a binary relation over these atoms. We will use this relation to indicate a partial order of *preferences* among disbelief atoms, and hence will refer to $>$ as the *preference relation*. Define a *preference set* P to be a set consisting of any number of instances of the $>$ relation, ie. $P = \{\neg L\sigma_i > \neg L\tau_i \mid i \in I\}$, for some index set I . An AE expansion T for a given A will be said to satisfy P iff for each $i \in I$ either:

1. $\neg L\tau_i \notin A$, or
2. σ_i is *legitimately derived* with respect to P and A

Legitimate derivation of σ_i with respect to P and A may be expressed informally as σ_i being a logical consequence of A and only those assumptions of disbelief atoms which are not lower in the preference ordering than $\neg L\sigma_i$.³

In essence, legitimate derivation of a formula σ_i with respect to a given set of preferences corresponds to the notion of σ_i not being derived from less-preferred assumptions of disbelief. For a given base set A of beliefs containing the defaults $\{\alpha_i(x) \wedge \neg L\phi_i(x) \supset \gamma_i(x) \mid i \in I\}$ this definition can be used to filter out unsanctioned expansions by establishing P as the set $P = \{\neg L\phi_i(x) > \neg L\gamma_i(x) \mid i \in I\}$.

C.6 Hierarchic defeat

The principle of hierarchic defeat can be expressed in AE logic by considering pairs of defaults. Let $\mathcal{D}_1 = A \rightarrow C_1$ and $\mathcal{D}_2 = B \rightarrow C_2$ be two defaults; for every such pair we add the schema:

³A more technical presentation of legitimate derivation is not possible without a lengthy digression into the mechanics of AE logics — see [54] for further details.

$$(A \subset B \wedge \neg(C_1(x) \wedge C_2(x))) \supset \phi_2(x) . \quad (3.7)$$

$\mathcal{D}_2$ will be defeated if A is a subset of B , and their conclusions conflict for some individual.

The schema (3.7) is unfortunately not a local translation of default statements, because it requires considering every pair of defaults. Still, it is almost local: it does not require looking at every default, or any of the “hard facts.”

D Conclusion

We have presented some parts of a theory of defaults, including a principle of adjudication under hierarchic inheritance. This theory can be expressed by an almost-local translation into autoepistemic logic, using a combination of epistemic concepts and the introduction of defeater predicates. The use of epistemic operators in the formalization seems necessary if we are to faithfully represent various types of defeat.

Chapter 4

REASONING ABOUT ACTIONS IN MULTIAGENT DOMAINS

This work was originally reported at the Workshop on Planning and Action in Timberline, Oregon in July 1986. It was written by Michael Georgeff.

Any approach to reasoning about and coordinating plans in dynamic, multiagent domains requires that we have available a model of action specifically suited to such domains. Classical models of action are not well-suited: in particular, they do not allow multiple actions (or events) to be performed simultaneously, except through the use of some interleaving approximation [90]. This considerably complicates reasoning about dynamic multiagent domains, especially where causality is involved.

During the most recent phase of this project we developed a model of action suited to multiagent domains that properly modeled the simultaneous performance of actions and events. The basis of this model is described below.

A Introduction

In developing automatic systems for planning and reasoning about actions, it is essential to have epistemologically adequate models of events, actions, and plans. Most early work in action planning assumed the presence of a single agent acting in a static world. In the formulation of these problems, the world was considered to be in one of a potentially infinite number of states and actions were viewed as mappings between these states [25,70,78,90]. However, the formalisms developed did not allow for simultaneous action, and as such are inadequate for dealing effectively with most real-world problems that involve other agents and dynamically changing environments.

Some attempts have recently been made to provide a better underlying theory of action. McDermott [80] considers an action (or event) to be a set of sequences of states and describes a temporal logic for reasoning about such actions. Allen [3] adopts a similar view and specifies an action by giving the relationships among the intervals over which the action's

conditions and effects are assumed to hold. Related formalisms have been developed by Dean [19], Pelavin [91] and Shoham [103].

A quite different and potentially powerful approach has recently been proposed by Lansky [61]. Instead of modeling actions and events in terms of world states, she regards events as primitive and defines states derivatively.

In this paper, we shall examine some of the problems that arise in the representation of events, actions, and plans in multiagent domains, and describe a model of events and actions that overcomes most of these problems.

B Actions and Events

We consider that, at any given instant, the world is in a particular *world state*. Each world state consists of a number of *objects* from a given domain, together with various *relations* and *functions* over those objects. A sequence of world states is called a *world history*.

A given world state has no duration; the only way the passage of time can be observed is through some change of state. The world changes its state by the occurrence of *events* or *actions*.¹ An *event type* is a set of state sequences, representing all possible occurrences of the event *in all possible situations* [3,80]. Except where the distinction is important, we shall call event types simply events.

We shall restrict our attention herein to *atomic events*. An atomic event is one in which each state sequence comprising the event contains exactly two elements; it can thus be modeled as a transition relation on world states. The transition relation of a given event must comprise all possible state transitions, *including those in which other events occur simultaneously with the given event*. Consequently, the transition relation of an atomic event places restrictions on those world relations that are directly affected by the event, but leaves most others to vary freely (depending upon what else is happening in the world). This is in contrast to the classical approach, which views an event as changing some world relations but leaving most of them unaltered.

For example, consider a domain consisting of blocks *A* and *B* at possible locations 0 and 1. Assume a world relation that represents the location of each of the blocks, denoted *loc*. Consider two events, *move(A, 1)*, which has the effect of moving block *A* to location 1, and *move(B, 1)*, which has a similar effect on block *B*. According to the classical approach [90], these events would be modeled as follows:

$$\begin{aligned} \text{move}(A, 1) = \{ & \langle \text{loc}(A, 0), \text{loc}(B, 1) \rangle \rightarrow \langle \text{loc}(A, 1), \text{loc}(B, 1) \rangle \\ & \langle \text{loc}(A, 0), \text{loc}(B, 0) \rangle \rightarrow \langle \text{loc}(A, 1), \text{loc}(B, 0) \rangle \} \end{aligned}$$

and similarly for *move(B, 1)*.

Every instance (transition) of *move(A, 1)* leaves the location of *B* unchanged, and similarly every instance of *move(B, 1)* leaves the location of *A* unchanged. Consequently, it is

¹From a technical standpoint, we shall use these terms synonymously.

impossible to compose these two events to form one that represents the simultaneous performance of both $move(A, 1)$ and $move(B, 1)$, except by using some interleaving approximation [90].

In contrast, our model of these events is

$$\begin{aligned} move(A, 1) = & \{ \langle loc(A, 0), loc(B, 1) \rangle \rightarrow \langle loc(A, 1), loc(B, 1) \rangle \\ & \langle loc(A, 0), loc(B, 1) \rangle \rightarrow \langle loc(A, 1), loc(B, 0) \rangle \\ & \langle loc(A, 0), loc(B, 0) \rangle \rightarrow \langle loc(A, 1), loc(B, 1) \rangle \\ & \langle loc(A, 0), loc(B, 0) \rangle \rightarrow \langle loc(A, 1), loc(B, 0) \rangle \} \end{aligned}$$

and similarly for $move(B, 1)$.

This model represents all possible occurrences of these events, including their simultaneous execution with other events. For example, if $move(A, 1)$ and $move(B, 1)$ are performed simultaneously, the resulting event will be the intersection of their possible behaviors:

$$\begin{aligned} move(A, 1) \parallel move(B, 1) &= move(A, 1) \cap move(B, 1) \\ &= \{ \langle loc(A, 0), loc(B, 0) \rangle \rightarrow \langle loc(A, 1), loc(B, 1) \rangle \} \end{aligned}$$

Thus, to say that an event has taken place is simply to place constraints on some world relations, while leaving most of them to vary freely.

Of course, to specify events by listing all the possible transitions explicitly would, in any interesting case, be infeasible. We therefore need some formalism for describing events and world histories. The one we use here is a generalization of the situation calculus [78], although most of our remarks would apply equally to other logic-based formalisms.

We first introduce the notion of a *fluent* [78], which is a function defined on world states. If we are using predicate calculus, the values of these fluents will range over the relations, functions, and objects of the domain. For example, the location of a given block A is a fluent whose value in a given state is the location of block A in that state. If, in a state s , the location of A is 1, we shall write this as $holds(loc(A, 1), s)$. Expressions denoting fluents that range over objects are often called *designators*, with a distinction drawn between those whose denotations are constant over all states (so-called *rigid* designators) and those whose denotations may vary (*nonrigid* designators).

As in the single-agent case, the well-formed formulas of this situation calculus may contain logical connectives and quantifiers; they can thus express general assertions about world histories. However, we do not use a “result” function to specify the state resulting from an event (or action). The reason is that, in our formalism, events are not functions on states but rather relations on states, and the occurrence of an event in a given state need not uniquely determine the resulting state. Therefore, for a given world history w containing state s , we let $succ(s, w)$ be the successor of s , and use a predicate $occurs(e, s)$ to mean that event e occurs in state s . This formulation, in addition to allowing a wider class of events than in the standard situation calculus, also enables us to state arbitrary temporal constraints on world histories [91].

In reasoning about actions and events, one of the most important things we need to know is how they affect the world – that is, we must be able to specify the effects of actions and events when performed in given situations. We can do this as follows.

Let ϕ and ψ be relational fluents. Then we can describe the effects of an event e with axioms of the following form:²

$$\forall w, s. \text{holds}(\phi, s) \wedge \text{occurs}(e, s) \supset \text{holds}(\psi, \text{succ}(s, w)) \quad (4.1)$$

This statement is intended to mean that, if ϕ is true when event e occurs, ψ will be true in the resulting state. It has essentially the same meaning as $\phi \supset [e]\psi$ in dynamic logic [41].

With axioms such as these, we can determine the strongest [provable] postconditions and weakest [provable] preconditions of arbitrary events and actions. These can then be used to form plans of action to accomplish given goals under prescribed initial conditions [73,99].

It is important to note that axioms of the above form cannot characterize the transition relation of any given event completely, no matter how many are provided. For example, with such axioms alone, it is not possible to prove for any two actions that they can be performed concurrently (nor that a plan containing concurrent actions is executable). We shall have more to say about this later.

Of course, we are often able to make stronger statements about actions and events than given above. For example, the event $\text{move}(A, 1)$ satisfies

$$\forall w, s. \text{occurs}(\text{move}(A, 1), s) \equiv \text{holds}(\text{loc}(A, 0), s) \wedge \text{holds}(\text{loc}(A, 1), \text{succ}(s, w))$$

This specification characterizes the event $\text{move}(A, 1)$ completely – there is nothing more that can be said about the event or, more accurately, about its associated transition relation. (The importance of this distinction will be made clear when we consider causal relationships among events.)

Thus, *at this point in the story*, the frame problem [42,78] does not arise. Because events, per se, need not place any restrictions on the majority of world relations, we do not require a large number of frame axioms stating what relations are left unchanged by the performance of an event (indeed, such statements would usually be false). In contrast to the classical approach, we therefore do not have to introduce any *frame rule* [42] or STRIPS-like assumption [25] regarding the *specification* of events.

C Independence

We have been regarding atomic actions or events as imposing certain constraints on the way the world changes while leaving other aspects of the situation free to vary as the environment chooses. That is, each action's transition relation describes all the potential

²We have simplified the notation in two ways. First, without stating so explicitly, we assume throughout that s and $\text{succ}(s, w)$ are elements of w . Second, we shall often use event *types* to stand for an event *instance* of the given type. Thus, axiom 4.1 should be viewed as shorthand for the following axiom, where ι is an event instance:

$$\forall w, s, \iota. \text{element}(s, w) \wedge \text{element}(\text{succ}(s, w), w) \wedge \text{type}(\iota, e) \wedge \text{holds}(\phi, s) \wedge \text{occurs}(\iota, s) \supset \text{holds}(\psi, \text{succ}(s, w))$$

changes of world state that could take place during the performance of the action. Which transition actually occurs in a given situation depends, in part, on the actions and events that take place in the environment. However, unless we can reason about what happens when some subset of all possible actions and events occurs – for example, when the only relevant actions being performed are those of the agent of interest – we could predict very little about the future and any useful planning would be impossible.

To handle this problem, we first introduce the concept of *independence*. We define a predicate $indep(p, e, s)$, which we take to mean that the fluent p is independent of (i.e., not directly affected by) event e in situation s . Unlike classical models of actions and events, this does not mean that, if we are in a state s in which p holds, p will also hold in the resulting state. Rather, if p is independent of e in some state s , the transition relation associated with e will include transitions to states in which p does not hold, as well as ones in which p holds, while not constraining the values of any other fluents in the resulting state.³ In the general case, we have to specify independence for all three types of fluents: the relation-valued, function-valued, and object-valued.

For example, we might have

$$\forall s, x, y. holds(x \neq A, s) \supset indep(loc(x, y), move(A, 1), s) .$$

This axiom states that, for all x and y , $loc(x, y)$ will be independent of event $move(A, 1)$, provided that x is not block A .

In our ontology, a world state can change only through the occurrence of events. Furthermore, in keeping with our intuitive notion of independence, events that are independent of some property cannot influence that property. We therefore have

$$\forall w, s, \phi. holds(\phi, s) \wedge \neg holds(\phi, succ(s, w)) \supset \exists e. (occurs(e, s) \wedge \neg indep(\phi, e, s)) \quad (4.2)$$

From this we can directly deduce the following *law of persistence*:

$$\forall w, s, \phi. holds(\phi, s) \wedge \forall e. (occurs(e, s) \supset indep(\phi, e, s)) \supset holds(\phi, succ(s, w)) \quad (4.3)$$

This rule states that, if we are in a state s where some condition ϕ holds, and if all events that occur in state s are independent of ϕ , then ϕ will also hold in the next (resulting) state. For example, we could use this rule to infer that, if $move(A, 1)$ were the only event to occur in some state s , the location of B would be the same in the resulting state as it was in s .

Unlike many other approaches to persistence [19,40 42,80,94,103], the foregoing law is *monotonic*; that is, the law does not involve any nonmonotonic operators or depend on any consistency arguments. Nor is it some fortuitous property of the world in which we live. Rather, it is a *direct consequence* of our notions of event and independence. What makes

³Independence can be defined as follows. Let $tr(e, s, s')$ denote that $\langle s, s' \rangle$ is an element of the transition relation associated with event e . Then we have that p is independent of e in state s if and only if, for all ϕ such that ϕ is consistent with both p and $\neg p$, $(\exists s'. tr(e, s, s') \wedge holds(p \wedge \phi, s')) \equiv (\exists s'. tr(e, s, s') \wedge holds(\neg p \wedge \phi, s'))$. This, of course, is not computable. Note also that an event transition relation may include states that cannot occur (because of some domain constraint) in any world history.

planning useful for survival is the fact that we can structure the world in a way that keeps most properties and events independent of one another, thus allowing us to reason about the future without complete knowledge of all the events that could possibly be occurring.

At first glance, however, it appears as though we would encounter considerable difficulty in specifying independence, simply on the grounds that it should be ascertainable for each possible fluent/event pair. Indeed, this is hardly surprising, as the foregoing law of persistence is little different in this respect from the original formalisms that gave rise to the frame problem [78].

There are two ways we could deal with this difficulty. One is to remain monotonic and rely on general axioms regarding independence to reduce combinatorial complexity. The other is to apply some nonmonotonic rule or minimization criterion that would allow independence to be specified more succinctly. I discuss the nonmonotonic approach elsewhere [33]; herein, I want to examine briefly the monotonic specification of independence, so as to show that such an approach is – in some cases – a practical alternative.

One way in which the combinatorics can be substantially reduced is by explicitly specifying *all* the events that could possibly affect each fluent.⁴ For example, for a given fluent p , we might have axioms such as the following:

$$\forall s, e. \neg indep(p, e, s) \supset ((e = e_1) \vee (e = e_2) \vee \dots)$$

or, in its contrapositive form

$$\forall s, e. \neg((e = e_1) \vee (e = e_2) \vee \dots) \supset indep(p, e, s)$$

As one would expect that, out of all possible events, there will be relatively few that affect a given fluent, such specifications can reduce considerably the combinatorics of providing separate independence axioms for each fluent/event pair. A minor complication is that, because we allow composite events (such as $e_1 || e_2$, and that combined with, say, e_3 , and so forth), the axioms for independence cannot be quite so simple as given above. However, this presents no serious difficulty.

A more substantial problem, however, is that this approach requires one to know the effects of *all* actions and events that could possibly occur. That is, such axioms do not allow for the possibility that *unspecified* events could affect the fluents of interest. This approach would therefore seem too strong for many real-world applications, though may be useful in less general contexts.

There are other ways to specify independence, however, that manage to avoid the combinatorial problem, yet do so without banishing unspecified events from the scene and without introducing nonmonotonicity. In particular, it may be possible to provide axioms describing the extent to which various actions and events exert their influence. For example, it may be that events outside a particular region R cannot affect properties inside that region:

$$\forall s, e, \phi. internal_f(\phi, R, s) \wedge external_e(e, R, s) \supset indep(\phi, e, s)$$

⁴This is essentially what Lansky does when she defines state predicates in terms of events [61].

In this way, a single axiom can specify independence for an entire class of fluent/event pairs. In large real-world domains this will invariably lead to a substantial reduction in the combinatorics of the problem. In small blocks worlds, on the other hand, it will not – but writing down all the independence axioms in such a case is not much of a problem either.

D Interference

If we are interested in constructing plans of action, one of the more important considerations is whether or not the actions constituting such plans are indeed performable. In single-agent planning, this question is quite easily handled by means of explicitly specifying preconditions that guarantee action performability. As we shall see, however, it is much more complex in multiagent domains.

The source of the problem in multiagent planning is that it is not possible to state simple preconditions for each individual action, the satisfaction of which would ensure its performability. In multiagent domains, whether or not an action can be performed will depend not only on the fulfillment of such preconditions, but also on which events or actions may (or are required to) occur simultaneously with the given action; it is, after all, of little use to form a plan that calls for the simultaneous or concurrent performance of actions that are inherently precluded from coexisting.

This problem is far more crucial than it may first appear. In particular, we are not concerned merely with issues of deadlock avoidance. In planning and other forms of practical reasoning, the failure of an action does not necessarily mean that the agent or device performing the action will thereafter be unable to proceed. Rather, such failure is usually taken to mean that the *desired* or *intended* effects of the action have not been achieved. Thus, though true deadlock may occur quite rarely, actions often fail to produce their intended effects because of interference with other, often unanticipated events.

Moreover, much of human planning revolves around the *coordination* of plans of action. Some of this is concerned with synchronizing the activities of agents so that tasks involving more than one agent can be carried out successfully. Such synchronization can be accomplished by specifying explicitly what temporal relations should hold among the activities of the various agents [61,111]. The more difficult problem is to identify interactions among potentially conflicting actions. Indeed, the recognition of possible plan conflicts is considered by some philosophers to be at the heart of rational behavior [8].

One way to specify such constraints on actions and events is to provide explicit axioms stating which events should occur simultaneously and which should not. For example, we could have the axiom

$$\forall s. \neg(\text{occurs}(e_1, s) \wedge \text{occurs}(e_2, s))$$

to mean that event e_1 could not occur simultaneously with event e_2 . This is exactly the approach employed by Lansky [61] and Pelavin [91]. However, while it seems that the synchronization of actions for cooperative tasks is most naturally expressed directly (that is, by explicitly specifying the required temporal relations between specific actions), it seems

unreasonable to require that all possible action *conflicts* also be so specified. For most real-world domains, it is more natural to specify just the effects of actions and to *deduce*, as the need arises, whether or not any two actions will interfere with each another. Furthermore, for domains of any complexity, there are potentially a very large number of actions and events that could interfere with one another. In such cases, the explicit specification of interference would entail severe combinatorial difficulties, although *appropriate* structuring of the problem domain [61] could substantially reduce the combinatorics.

It is desirable, therefore, to be able to determine freedom from conflict for any specified events, given simply a description of the effects of these events upon the world. To do this, we need to prove that the intersection of the transition relations corresponding to the events of interest is nonempty.

At first glance, it appears as if axioms about the effects of events are *all* we really need for determining the possibility or not of event simultaneity. For example, let us assume we have the following axioms describing events e_1 and e_2 :

$$\forall w, s . \text{holds}(p, s) \wedge \text{occurs}(e_1, s) \supset \text{holds}(q_1, \text{succ}(s, w))$$

$$\forall w, s . \text{holds}(p, s) \wedge \text{occurs}(e_2, s) \supset \text{holds}(q_2, \text{succ}(s, w))$$

From this we can infer that

$$\forall w, s . \text{holds}(p, s) \wedge \text{occurs}(e_1 || e_2, s) \supset \text{holds}(q_1 \wedge q_2, \text{succ}(s, w))$$

Nevertheless, it would be unwise to take these axioms as the basis of a plan to achieve $q_1 \wedge q_2$. The reason is that, given these axioms alone (or any others of the same form), it is simply not possible to *prove* that events e_1 and e_2 can occur simultaneously. Nor is it possible to use consistency arguments to justify the assertion that these events can so occur; indeed, whether or not these events can take place simultaneously depends on how they affect other world properties. For example, simultaneity would be impossible if e_1 , say, always resulted in r being true while e_2 always resulted in r being false. (Of course, given *sufficient* axioms about the effects of e_1 and e_2 , we could, in this case, prove that they *could* not occur together.)

Even if we are given necessary and sufficient conditions for the occurrence of events [3], we are still not out of the woods. For example, consider that events e_1 and e_2 satisfy the following axioms:

$$\forall w, s . \text{occurs}(e_1, s) \equiv \text{holds}(p, s) \wedge \text{holds}(q_1, \text{succ}(s, w))$$

$$\forall w, s . \text{occurs}(e_2, s) \equiv \text{holds}(p, s) \wedge \text{holds}(q_2, \text{succ}(s, w))$$

That is, a necessary and sufficient condition for e_1 having occurred is that p holds at its inception and q_1 holds at its completion; for e_2 , q_2 must hold at its completion. But, even in this case, the best we can do is to try to prove that it is *consistent* for these events to occur simultaneously. This is clearly unsatisfactory from a computational standpoint. Furthermore, such reasoning is essentially nonmonotonic; the addition of further axioms

may render previously consistent formulas inconsistent and any previous conclusions about possible event simultaneity may have to be withdrawn.

Another alternative is to determine interference by checking mutual independence for every fluent in the domain [32]. The major problem with such an approach is that this determination has to be made for every possible fluent, *including unspecified ones*. For example, despite the fact that two events may be mutually independent with respect to every specified fluent in the domain, there may exist some unspecified fluent for which they are not mutually independent. We are thus required to assume that the explicitly denoted fluents are the *only* ones relevant to the determination of interference.

Just as we wanted to avoid introducing any nonmonotonic operator or consistency criterion into our law of persistence, here also we want to avoid any form of nonmonotonicity. The solution we propose is based on being able to specify conditions under which we can guarantee performability of a given action or event. Such a condition will be called a *correctness condition* and, for a given event e , condition p , and state s , will be denoted $cc(p, e, s)$. The intended meaning of this statement is that any event that does not interfere with (affect) condition p will not interfere with (prevent) the occurrence of event e .⁵ In addition, of course, we would need appropriate axioms defining the preconditions for the performance of e , but this is easily handled in the standard manner.

We now introduce the notion of freedom from interference. We shall say that events e_1 and e_2 are *interference-free* in a state s if the following condition holds:

$$\exists \phi, \psi . cc(\phi, e_1, s) \wedge cc(\psi, e_2, s) \wedge indep(\phi, e_2, s) \wedge indep(\psi, e_1, s)$$

This condition will hold if, in state s , events e_1 and e_2 have no direct effect on the same properties of the domain. For example, consider the events $move(A, 1)$ and $move(B, 1)$ described earlier. We have the following:⁶

$$\forall s, x, y, X . holds(x \neq X, s) \supset indep(loc(x, y), move(X, 1), s)$$

$$\forall s, X . cc(loc(X, 1), move(X, 1), s)$$

If A and B are assumed to denote different objects, it is easy to see that $move(A, 1)$ and $move(B, 1)$ are interference-free. Note that we have assumed that both A and B can occupy the same location at the same time. If this were not the case, the correctness conditions for $move(A, 1)$ and $move(B, 1)$ would have to be altered to include this additional constraint. The events would then not be interference-free.

It immediately follows that two events will be able to occur simultaneously in a state s if

1. The preconditions of each event are satisfied in s , and

⁵Correctness conditions can be defined as follows. Let $tr(e, s, s')$ denote that $\langle s, s' \rangle$ is an element of the transition relation associated with event e . Then we have that p is a correctness condition for an event e if and only if, for all ϕ such that both ϕ and $\neg\phi$ are consistent with p , $(\exists s' . tr(e, s, s') \wedge holds(p \wedge \phi, s')) \equiv (\exists s' . tr(e, s, s') \wedge holds(p \wedge \neg\phi, s'))$. As with the definition of independence, this is not computable.

⁶We assume x and y are rigid designators; see reference [73] for a discussion of this issue.

2. The events are interference-free in s .

It should be noted that, if we wish to show that two events can proceed concurrently (which not only includes simultaneity but also allows either event to precede the other), we also need to prove that the preconditions of each event are independent of the other event. This can be done in the same way as for the correctness conditions. One might, in such circumstances, reserve the term "interference-free" for events that affect neither each other's correctness conditions nor preconditions [29]. Actions $move(A, 1)$ and $move(B, 1)$ are also interference-free in this stronger sense, as neither action affects the preconditions of the other.

In case two events do affect the same fluents (and thus do not satisfy the condition of interference freedom given above), it might yet be that the fluents are affected in the same way. If this is so, we say that the events are *compatible*. Compatible events can occur simultaneously, and in this sense are also interference-free.

E Causality

One problem that we have not properly addressed is the apparent complexity of the axioms of independence and correctness. For example, while it might seem reasonable to state that the location of block B is independent of the movement of block A , as everyone knows, this is simply untrue in most interesting worlds. Whether or not the location of B is independent of the movement of A will depend on a whole host of conditions, such as whether B is in front of A , on top of A , on top of A but tied to a door, and so on. Indeed, it is often this apparent endless complexity rather than the combinatorial factors that many people have in mind when they refer to the frame problem.

One way to solve this problem is by introducing a notion of *causality*. We allow two kinds of causation, one in which an event causes the simultaneous occurrence of another event, and the other in which an event causes the occurrence of a consecutive event. We denote these two causal relations by $causes_s(\phi, e_1, e_2)$ and $causes_n(\phi, e_1, e_2)$, respectively, where ϕ is the condition under which event e_1 causes event e_2 . These two kinds of causality are sufficient to describe the behavior of any procedure, process, or device that is based on discrete (rather than continuous) events.

The axioms expressing the effects of causation are

$$\forall w, s, \phi, e_1, e_2 . causes_s(\phi, e_1, e_2) \wedge holds(\phi, s) \wedge occurs(e_1, s) \supset occurs(e_2, s)$$

$$\forall w, s, \phi, e_1, e_2 . causes_n(\phi, e_1, e_2) \wedge holds(\phi, s) \wedge occurs(e_1, s) \supset occurs(e_2, succ(s, w))$$

For example, we might have a causal law to express the fact that, whenever a block x is moved, any block on top of x and not somehow restrained (e.g., by a string tied to a door) will also move. We could write this as

$$\forall x, y, l . causes_s((on(y, x) \wedge \neg restrained(y)), move(x, l), (move(y, l)))$$

If this axiom holds, the movement of x will *cause* the simultaneous movement of y whenever y is on top of x and is not restrained.

We use the notion of causality in a purely technical sense and, while it has many similarities to commonsense usage, we are not proposing it as a fully-fledged theory of causality. Essentially, we view causation as a relation between atomic events that is conditional on the state of the world. We also relate causation to the temporal ordering of events, and assume that an event cannot cause another event that precedes it. However, as stated above, we do allow an event to cause another that occurs simultaneously. This differs from most other formal models of causality [61,80,103], although Allen [3] also allows simultaneous causation.

We can also use causality to maintain invariants over world states and to simplify the specification of actions and events. Consider, for example, a seesaw, with ends A and B and fulcrum F (Figure 1). Assume that A , F , and B are initially at location 0, and consider an event $move_F$ that moves F to location 1. Because of the squareness of the fulcrum and the constraint that A , F , and B must always remain collinear, $move_F$ also results in the movement of A and B to location 1.

We could model $move_F$ by a somewhat complex event that, in and of itself, would affect not only the location of F , but also the locations of A and B . Using this model, the locations of F , A , and B *would not* be independent of $move_F$. However, an alternative view would be to consider that the only property affected by $move_F$ is the location of the fulcrum F , and to let $move_F$ *cause* the simultaneous movement of A and B .

For example, we might have the following causal laws:

$causes_s(true, move_F, move(A, 1))$

$causes_s(true, move_F, move(B, 1))$

The intended meaning of these causal laws is that, if we perform the event $move_F$, both $move(A, 1)$ and $move(B, 1)$ are caused to occur simultaneously with $move_F$.

With this axiomatization, the locations of A and B *will* be independent of $move_F$. Of course, because $move_F$ always causes the movement of A and B , their locations will always be *indirectly* affected by the movement of F ; however, from a technical standpoint, we consider the locations of A and B independent of $move_F$ itself (though not of the composite event that consists of the simultaneous movements of F , A , and B). The axioms of independence (and, similarly, correctness) can thus be considerably simplified, but at the cost of introducing more complex causal laws. The advantage of doing things this way is that the complexity is thereby shifted to reasoning about relationships among events, but away from reasoning about the relationships between events and fluents.

There are a number of things to be observed about this approach. First, provided that we add a domain constraint requiring that A , F , and B always remain collinear, we could simplify the above causal laws so that, for example, we require only that $move(A, 1)$ be caused by $move_F$. Using the collinearity constraint, together with the law of persistence, we can then infer that there must exist yet another event that occurs in state s and that brings about the simultaneous movement of B .

In many cases, therefore, we do not need to include causal laws to maintain invariant world conditions; we can instead make use of the constraints on world state to infer the existence of the appropriate events. However, if we do adopt this approach, we shall have to find some way of inferring (either monotonically or nonmonotonically) which of all the potentially appropriate events is the intended one (for example, did the event move B alone, or did it have other effects on the world as well?).

Second, causal laws can be quite complex, and may depend on whether *or not* other events take place as well as on conditions that hold in the world. As a consequence, the application of causal laws need not yield a unique set of caused events. For example, one causal law could require that an event e_1 occur if e_2 does not, whereas another could require that e_2 occur as long as e_1 does not. Given only this knowledge of the world, the most we could infer would be that just one of the events has occurred – but *which* one would be unknown.

Third, for planning possible future courses of action, the extent of causality must somehow be limited. Unless we have either first-order axioms or some nonmonotonic rule to limit causation, any given event could conceivably cause the occurrence of any other event. Clearly, with the possibility of so many events occurring, any useful planning about the future becomes impossible. This is not a difficult problem, but some care must be taken in addressing it [33].

It is interesting to note that the “deductive operators” used in SIPE [116] are very much like the causal laws described herein. Furthermore, SIPE limits the extent of causation by use of an implicit closed-world assumption so that, if causation between any two events cannot be proved, it is assumed that no such relation exists. This yields precisely those events that are *causally necessary*.

Finally, some predicates are better considered as *defined*, which avoids overpopulating the world with causal laws. For example, the distance between two objects may be considered a defined predicate. Instead of introducing various causal laws stating how this relation is altered by various move events, we can simply work with the basic entities of the problem domain and infer the value of the predicate from its definiens when needed.

F Events as Behaviors

So far, we have identified actions and events with the sets of all their possible behaviors. However, at this point we encounter a serious deficiency in this approach, as well as all others that model actions and events in this way [3,19,80,91,102]. Let us return to the example of the seesaw described in the previous section. Again assume that A , F , and B are initially at location 0, and consider two actions, $move_F$ and $move'_F$, both of which move F to location 1 (see Figure 1). We require that both events also allow all possible movements of A and B , depending on what other events are occurring at the same time (such as someone lifting B). Of course, the objects must always remain collinear. However, $move_F$ and $move'_F$ differ in that, *when performed in isolation*, $move_F$ results in the simultaneous movement of A and B to location 1, whereas $move'_F$ leaves the location of A unchanged while moving B to location 2.

Because both $move_F$ and $move'_F$ exhibit exactly the same class of possible behaviors, the transition relations associated with each of these events will be identical. From a purely behavioral point of view, this is how things should be. To an external observer, it would appear that $move_F$, say, sometimes changed the location of A and not B (when some simultaneous event occurred that raised A to location 2), sometimes changed the location of B and not A (when some simultaneous event raised B), and sometimes changed the locations of both A and B . (Of course, $move_F$ would always change the location of F). As there is no *observation* that could allow the observer to detect whether or not another event was occurring simultaneously, there is no way $move_F$ could be distinguished from $move'_F$ or any other event that had the same transition relation.

On the other hand, it is very convenient to be able to make such distinctions; humans seem to have no trouble reasoning about events of this kind, and it would be unwise to exclude them from our theory. For example, $move_F$ and $move'_F$ may correspond to two different ways of moving F . On the surface, these events would appear to allow the same class of behaviors but, because of unobserved differences in the ways that they are performed, could exhibit different behaviors in specific situations. In other cases, while an event like $move_F$ might be appropriate to seesaws, an event isomorphic to $move'_F$ might be necessary for describing the movement of objects in other contexts. For example, consider the case in which, instead of being components of a seesaw, A is a source of light and B is F 's shadow.

Thus, if we wish to be able to distinguish events such as $move_F$ and $move'_F$, we have to allow events with identical transition relations to have differing effects on the world, depending on what other events are, or are not, occurring at the same time. Unfortunately, our current model of events simply leaves us no way to represent this. We cannot restrict the transition relation of $move_F$, say, so that it will always yield the state in which A , F , and B are all at location 1, because that would prevent A or B from being moved simultaneously to other locations. Similarly, we cannot restrict the transition relation of $move'_F$ to allow only the movement of B . Nor can we use any general default rule or minimality criteria to determine the intended effects of an event when performed in any specific context (because that would yield identical models for both $move_F$ and $move'_F$). Indeed, in the situation in which $move_F$ is performed in isolation, note that we do *not* minimize the changes in world relations or maximize their persistence: *both* A and B change location along with F .

We therefore consider events to be objects of the domain that have an associated transition relation, but do not require that events with the same transition relation be deemed equivalent. Events having the same transition relation may differ in other properties: in particular, they may play different causal roles in a theory of the world.⁷

For example, in the case of the seesaw, we might have the following axiom for $move_F$:

$causes_s(p, move_F, move(A, 1))$

where $p \equiv \lambda(s)(\forall e . occurs(e, s) \supset int-free(e, move(A, 1), s))$.

⁷In a previous paper [32] I identified events with transition relations and let actions alone assume different causal roles independently of their associated transition relation. Considering the vast literature that already exists on events and actions, and which, if anything, makes a somewhat different distinction, I now believe this to have been a bad idea.

The intended meaning of this causal law is that the movement of F will cause the movement of A to location 1, provided no event occurs that interferes with the movement of A . If desired, one could use a similar causal law to describe the movement of B .

On the other hand, $move'_F$ would cause the movement of B to location 2 when performed in isolation:

$$causes_s(q, move'_F, move(B, 2))$$

$$\text{where } q \equiv \lambda(s)(\forall e . occurs(e, s) \supset int-free(e, move(B, 2), s)) .$$

It is important to note that this view of events (and any view that, in one way or another, associates an event with the set of all its possible behaviors [3,19,80,91,102]) requires that the event transition relation include *all* possible transitions under all possible situations, *including* the simultaneous occurrence of other events. For example, if we wish to allow for two pushing events opposed to one another to exert sufficient frictional force on a block to enable it to be lifted, this composite event must be one of the permissible transitions in each of the individual push events. Thus, two events cannot be combined to yield, synergistically, an event that is not part of the actions themselves. It follows that, in the specification of an event, we can only state what is true of *all* possible occurrences of the event. In the case of the push event, it would be a mistake to write an axiom stating that the effect of a push event on a block is to move the block. While this may be true if there is but a single event affecting the block in question, it is clearly false when two or more such events are acting upon the block simultaneously.

What is true of all push events (and hence can be stated as an axiom) is that they exert a force on the object being pushed. Other axioms could then be used to specify under what conditions opposing forces generate sufficient frictional force to lift objects, while a third group of axioms could describe how the resultant forces on an object cause it to move. Depending on the desired level of description, this axiomatization could be either simplified or elaborated.

G Processes

It is often convenient to be able to reason about groups of causally interrelated events as single entities. For example, we might want to amalgamate the actions and events that constitute the internal workings of a robot, or those that pertain to each component in a complex physical system. Such groupings of events, together with the causal laws that relate them to one another, will be called *processes*.

We assume that we have a set of processes and can classify various events and fluents as being either internal or external with respect to these processes. Let $internal_f(\phi, P, s)$ and $external_f(\phi, P, s)$ denote these relationships as they hold between a fluent ϕ and process P in state s , and let $internal_e(e, P, s)$ and $external_e(e, P, s)$ denote these relationships between an event e and process P in state s .

We place a number of constraints on the internal and external fluents and events of a process. First, we require that, for both fluents and events, those classified as internal be

mutually exclusive of those classified as external. Second, we require that, in all situations, each internal event have a correctness condition that is dependent only on internal fluents. This property can be expressed as follows:

$$\forall e, s, P . \text{internal}_e(e, P, s) \supset \exists \phi . \text{internal}_f(\phi, P, s) \wedge cc(\phi, e, s)$$

We impose a similar constraint on the preconditions of internal events. Next, we require that internal fluents be independent of all events except internal ones:

$$\forall e, s, \phi, P . \text{internal}_f(\phi, P, s) \wedge \neg \text{internal}_e(e, P, s) \supset \text{indep}(\phi, e, s)$$

External events and fluents are required to obey similar constraints. It is then not difficult to prove that, if the above axioms are satisfied, the internal events and external events of a given process are interference-free.

Finally, we require that there be no *direct* causal relationship between internal and external events. Thus, the only way the internal events of a given process can influence the external events of the process (or vice versa) is through indirect causation by an event that belongs to neither category. Within concurrency theory, these intermediary events (more accurately, event types) are often called *ports*. Processes thus impose causal boundaries and independence properties on a problem domain, and can thereby substantially reduce combinatorial complexity. Lansky's notion of a *group* [61] is quite similar to our notion of process.

The ease with which processes can be identified will depend strongly on the problem domain. In standard programming systems (at least those that are well structured), processes can be used to represent scope rules and are fairly straightforward to specify. On NASA's proposed space station, most of the properties of one subsystem (such as the attitude control system) will be independent of the majority of actions performed by other subsystems (such as the environmental and life support system), and thus these subsystems naturally correspond to processes as defined here. Lansky [61] gives other examples in which processes are readily specified.

In other situations, the specification of processes might be more complicated. For example, we might know that interference at a distance can occur only as a result of electromagnetic or gravitational phenomena, and so utilize this knowledge to impose causal boundaries whenever electromagnetic emissions are shielded and gravitational forces are negligible. Moreover, in many real-world situations, dependencies will vary as the spheres of influence and the potential for interaction change over time. For example, consider how many actions in real life are taken solely for the purpose of limiting or enhancing interference with other systems (such as closing a door for privacy, camouflaging military equipment, or making a phone call).

If we are to exploit the notion of process effectively, it is important to define various composition operators and to show how properties of the behaviors of the composite processes can be determined from the behaviors of the individual processes. For example, we should be able to write down descriptions of the behaviors of individual agents, and from these descriptions deduce properties of groups of agents acting together (concurrently). We should *not* have to consider the internal behaviors of each of these agents to determine how

the group as a whole behaves. In contrast, all the so-called hierarchical planning systems (e.g., NOAH [100] and SIPE [116]) analyze interaction down to the atomic level (as was noticed early by Rosenschein [99]).

The existing literature on concurrency theory [47,81] provides a number of useful composition operators. Some examples are given below. They can all be defined in terms of the causal relations introduced earlier, although we need to introduce a special "no-op" or "wait" event to prevent forcing the processes to operate in lockstep with one another.

Prefixing (:)

The process $e : P$ is one that can begin by performing the event (strictly speaking, event type) e , after which it behaves exactly like P .

Sequencing (;)

The process $P ; Q$ behaves first like P and, if that concludes successfully, behaves next like Q .

Ambiguity (+)

The process $P + Q$ can behave like either P or Q . For example, if

$$R = (b : P) + (c : Q) \quad ,$$

then R can either perform b and evolve into P or perform c and evolve into Q .

Parallelism (&)

The process $P \& Q$ is one in which both P and Q run concurrently. Events that are designated as synchronous must occur simultaneously, whereas other events can choose to occur simultaneously with one another or be arbitrarily interleaved.

We must, of course, provide various axioms about these operators so that they will be useful. For example, it is not difficult to show that any [temporal] property that holds of the behaviors of a process P (or Q) will also hold for the composite process $P \& Q$.

Such axioms may not appear to be very useful, as the properties that hold of each process will, in general, depend on what other events could occur in the environment. However, to the extent that the properties of a process's behaviors are specified in terms of its internal events and fluents, they will be independent of the context in which the process is embedded. For example, consider the two very simple processes P and Q given below:

$$P = a : b : P, \text{ and}$$

$$Q = c : d : Q .$$

Strictly speaking, we have to provide a fixed-point operator to define these processes [47], but the intended meaning should be clear. Now, if the event types a and b are mutually exclusive and are internal with respect to process P , all behaviors of P will be such that the number of events of type a , denoted $\#a$, and the number of events of type b , $\#b$, obey the constraint: $\#a - 1 \leq \#b \leq \#a$. This would *not* be the case if a and b were not internal events of process P , as any other process could then choose to perform an arbitrary number of events of type a or b concurrently with process P .

Now let's assume that c and d are internal events of process Q , and thus obey a law similar to that given for P . Furthermore, let us assume that events of type b and d are always constrained to be simultaneous with some interface event e , and vice versa. Using the fact that the behaviors of the composite process $P \& Q$ will satisfy the same constraints as the component processes, it is not difficult to prove that $\#a - 1 \leq \#c \leq \#a + 1$.

Despite the triviality of the example, the important point is that, in proving the above result, we have not had to examine the internal workings of either process P or Q . For example, had the internal structure of these processes been entirely different, this result would have remained valid (provided, of course, that the processes had still imposed the same constraints on the number of occurrences of events a , b , c , and d). Furthermore, in determining the properties of the individual behaviors of P and Q we have not had to consider the external environment in which they are embedded; the relationship between the number or occurrences of events a and b in process P is independent of external happenings, and similarly for events c and d in process Q .

There are a number of complexities in the specification of processes that require some care. For example, a process may *fail* at any time (because some precondition has not been or cannot be met, or some correctness condition has been violated, etc.). Thus, the behaviors generated by a process will include both failed and successful behaviors, where each failed behavior is a prefix of some successful behavior. Indeed, the notion of a process generating both successful and failed behaviors is essential to planning in real-world domains – we frequently select a plan of action according to how it can *fail* rather than how it succeeds. Amy Lansky and I have discussed this question in more detail elsewhere [35].

Nondeterministic processes present another problem that has to be addressed with caution. Such processes can differ from one another despite the fact that they generate identical behaviors (both successful and failed). The reason for this is that nondeterministic processes can behave differently in different environments even though their sets of potential behaviors are identical. This issue is explored at length in the literature on concurrency and various means of handling the problem have been developed [47,81].

In summary, the notion of process allows us to structure problem domains and thus avoid considering how every event affects every other event or fluent. It can therefore lead to a substantial reduction in the combinatorics of the problem. Furthermore, we can describe complex domains in a compositional way – that is, we can specify properties of individual processes independently of other processes, and then determine the behavior of the combined processes from knowledge of the behaviors of the component processes.

H The Frame Problem

The frame problem, as Hayes [42] describes it, is concerned with providing, in a reasonably natural and tractable way, appropriate “laws of motion.” Our approach to this problem has been to provide a proper model-theoretic account of actions and events, and to formulate first-order axioms and rules that allow the effects of actions and events to be determined monotonically. Furthermore, by introducing the notions of independence and correctness,

we obviate the necessity of using nonmonotonic operators or consistency arguments to obtain useful results regarding persistence and interference.

Of course, we are left with the problem of *specifying* independence and correctness. There are essentially two problems here: (1) the apparent combinatorial difficulties in expressing all the required independence and correctness axioms; and (2) the complexity to be expected of many, if not most, of these axioms in real-world applications.

The first of these problems is probably overstated in much of the literature on the frame problem. Thus, just as in the axiomatization of any large or infinite domain, the number of axioms required to specify independence (and correctness) can be substantially reduced by the use of general axioms that allow specific instances of independence to be deduced as needed. This can substantially reduce the combinatorial problem and has the advantage of remaining within the bounds of first-order logic.

Where desired, some simple closed-world assumption or minimality criterion could additionally be used to ease specification of the independence relation for the basic fluents of the domain. This can be quite straightforward, although some care has to be taken with the situation variable [33,40].

The second problem is handled by introducing causal laws that describe how actions and events bring about (cause) others. Indeed, without such causal laws, the specification of independence would become as problematic as is the specification of persistence in the standard formalisms. Of course, the causal laws can themselves be complex (just as is the physics of the real world), but the representation and specification of actions and events are thereby kept simple.

Some researchers take a more general view of the frame problem, seeing it as the problem of reasoning about the effects of actions and events with *incomplete information* about what other events or processes (causally related or otherwise) may also be occurring. Unfortunately, this problem is often confused with that of providing an adequate *model* of events, with the result that there is usually no clear model-theoretic semantics for the representation.

For example, one of the major problems in reasoning about actions and plans is to determine which events can possibly occur at any given moment. Based on the relative infrequency of "relevant" events (or that one would "know about" these if they occurred), it has been common to use various default rules [94], nonmonotonic operators [19,80], or minimal models [69,75,103] to constrain the set of possible event occurrences. However, there are many cases in which this is unnecessary – where we can *prove*, on the basis of general axioms about independence and correctness, that no events (or effects) of interest could possibly occur. We may even have axioms that allow one to avoid considering whole classes of events, such as when one knows that certain events are external with respect to a given process. Thus, in many cases, there may be no need to use default rules or minimality principles – reasoning about plans and actions does not have to be nonmonotonic.

When we *do* need to make assumptions about event occurrences, default rules and circumscription can be very useful. For example, by minimizing the extensions of the *occurs* and *causes* predicates, we can obtain a theory in which the only event occurrences

are those that are *causally necessary* [33]. This kind of reasoning seems to correspond closely to much of commonsense planning. However, in making reasonable assumptions about a given domain, we do not have to limit ourselves to such default rules or minimality criteria. In some cases, it may be preferable to use, for example, domain-specific rules defining what assumptions are appropriate or, alternatively, a more complicated information-theoretic approach based on quantitative probabilities of event occurrences.

To take a familiar example [3], it seems reasonable, at first, to assume that my car is still where I left it this morning, unless I have information that is inconsistent with that assumption. However, this premise gets less and less reasonable as hours turn into days, weeks, months, years, and centuries – even if it is quite *consistent* to make such a premise. This puts

the problem where it should be – namely, in the area of making reasonable assumptions, not in the area of *defining* the effects of actions [25,42,94], the persistency of facts [19,80], or causal laws [103].

Of course, most of the axioms we might state about the real world are subject to *qualification* [78]. Our aim has not been to solve this problem, although the notion of causality helps to some extent. Furthermore, because our approach has a well-defined semantics and can be formalized in first-order logic, we provide a sound base on top of which can be built various meta-theories regarding the handling of qualifications and other kinds of assumptions. I address some of these issues elsewhere [33].

I Conclusions

We have constructed a model of actions and events suited to reasoning about domains that involve multiple agents or dynamic environments. The proposed model provides for simultaneous actions, and a generalized situation calculus is used to describe the effects of actions in multiagent settings. Notions of *independence* and *correctness* were introduced and it was shown how they can be used to determine the persistence of facts over time and whether or not actions can be performed concurrently. Both these notions I consider critical to reasoning *effectively* about multiagent domains. Furthermore, unlike most previous formalisms in both single and multiagent domains, the proposed law of persistence is *monotonic* and therefore has a well-defined model-theoretic semantics.

We have also demonstrated how the concept of *causality* can be used to simplify the description of actions and to model arbitrarily complex machines and physical devices. It was also shown how sets of causally interrelated actions can be grouped together in *processes* and how this structuring of a problem domain can substantially reduce combinatorial complexity. The notion of structuring the problem domain by using general axioms about independence and causal influence appears to be essential for solving complex multiagent problems. The only existing work I know of that incorporates such an idea is the planning system being developed by Lansky [61].

Although we did not consider implementation issues directly, the concepts and laws introduced by us were aimed at providing a sound basis for practical planning and reasoning

systems. For example, one of the most efficient action representations so far employed in AI planning systems – the STRIPS representation [25,69] – is essentially the special case in which (1) the effects of an action can be represented by a conjunction of either positive or negative literals, called the *add list*; (2) the action is independent of all properties except those given in (or deducible from) the *delete list* of the action; (3) a single precondition determines performability of the action; and (4) no actions ever occur simultaneously with any other. The approach used by Pednault [90] can also be considered the special case in which there are no simultaneous actions.

Furthermore, the work here indicates how the STRIPS representation could be extended to the multiagent domain. For example, one possibility would be to stay with the single-agent representation, adding to it the requirement that the correctness conditions of an action be represented by the same conjunction of literals given in the add list of the action. Causal laws could be introduced in the manner of the deductive operators of SIPE [116], thereby increasing the expressive power of the approach without introducing the problems usually associated with extending the STRIPS assumption [94]. In this way, some of the traditional planning systems may be able to be modified to handle multiagent domains. Alternatively, the approach of Manna and Waldinger [73] could be applied to these domains by employing the generalized situation calculus we introduced here. Finally, our approach has shown how we can deduce constraints on the occurrence of actions and events from relatively simple axioms about their effects and influence. These constraints can then be used by event-based planners [28,61,111] to form synchronized plans involving the cooperation of multiple agents in dynamically changing environments.

I.1 Acknowledgments

I wish especially to thank Amy Lansky and Ed Pednault, both of whom helped greatly in clarifying many of the ideas presented in this paper. I am also indebted to Vladimir Lifschitz for some very enlightening discussions, and to Richard Waldinger who criticized an earlier draft of this paper.

Chapter 5

EVENT-BASED PLANNING FOR MULTIAGENT DOMAINS

This work was originally reported at the Workshop on Planning and Action in Timberline, Oregon in July 1986. It was written by Amy Lansky.

A Introduction

The duality between events and states is a well-known phenomenon. In a state-based representation, the world is viewed as a series of states or “snapshots” that are altered by events. Events are modeled solely in terms of their state-changing function. Alternatively, the dual, event-based approach represents the world in terms of a set of interrelated events. In this context, the “state” of the world at any particular point in time is represented in terms of the set of events that have occurred up to that moment (see Figure 5.1).

Most AI domain representations have relied on state-based models. In this paper we explore the dual view and examine its impact on the representation of multiagent domains — domains in which parallel activity is inherent and vital. As with most dualities, the choice of one representation over another may not affect any essential capability for expression; after all, one dual representation can usually be converted into the other. However, the mere form of a representation may make certain kinds of properties more natural to express and reason about. We believe that an event-based approach holds this advantage with respect to many of the complicated properties of multiagent worlds.

The representation described in this paper is based on the GEM concurrency model [63,64,65,66]. As advocated by philosophers such as Davidson [17] and as manifested in several AI representations such as Allen’s and Georgeff’s [3,32], GEM reifies events and explicitly represents their causal and temporal interrelationships. However, unlike previous AI representations, events are the *primary* elements of our world model and state is defined strictly in terms of past event activity. Thus, the work described in this paper explores the use of events and event relationships in way that is more general than previous work on knowledge representation and reasoning.

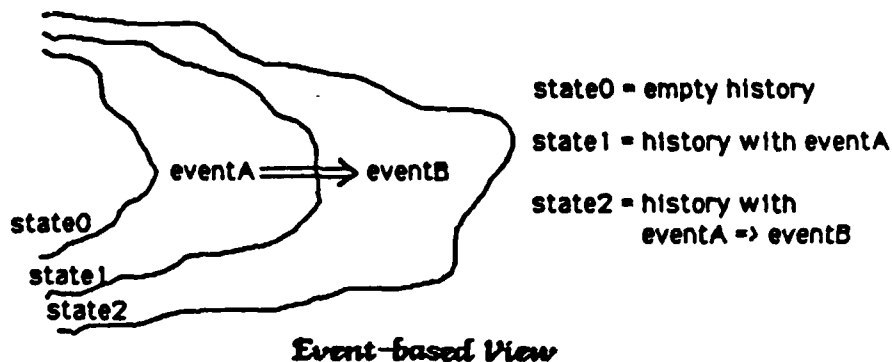
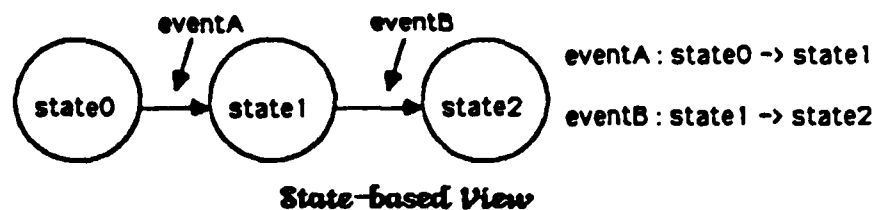


Figure 5.1: State/Event Duality

Another important aspect of the GEM representation is an explicit emphasis on *location* of activity. Specific mechanisms are provided for structuring events into logical locations of occurrence as well as for grouping those locations together in various ways. These event structures help organize the way a domain is described – for instance, particular domain constraints can be localized within a given context or subset of events. Structural contexts can also be used to actually represent properties of the domain. For example, particular event sets can be used to represent locations of forced sequential activity, scopes of potential causal effect, or the boundaries of localized forms of knowledge. In this way, domain structure helps to attack aspects of the frame problem. Domain structure can also be utilized as a heuristic in guiding computation; for example, it can serve as a guideline for the decomposition of planning tasks.

Within an event-based model, such as the one we are proposing, a notion of "state" is most naturally defined in terms of past activity: the state of the world at any point in time is merely a record of the events that have occurred and their interrelationships (once again, see Figure 5.1). *State descriptions* can be used to characterize sets of states, just as in a state-based model. These descriptions are typically formulas that describe patterns of past activity – for example, "the state in which a red robot has registered a request for tool X followed by a request for tool Y, but has not yet received either tool" or "the state in which Jack and Jill have just simultaneously begun running down the hill." These "behavioral" descriptions of state are formulas that explicitly describe temporal and causal relationships *between events*. In GEM, behavioral state descriptions (as well as all domain constraints) are stated as first-order temporal-logic formulas. Because these formulas are cast directly in terms of events and event relationships, a wide range of behavior-related properties can be described succinctly. Indeed, descriptions of states in which simultaneous activity has occurred are impossible to formulate in many state-based representations. As we will show, more conventional state descriptions based on state predicates can also be utilized in our event-based framework (see Section F).

The primary aim of this paper is to convey the expressive power behind a world model based on structured, interrelated events. We begin in Section B by motivating our underlying approach and relating it to other work. Section C provides a more formal description of the GEM world model and presents a semantics for our domain descriptions. The power behind this representation is then more fully illustrated in Section D through construction of a complex domain description. Finally, Sections E and F focus on the modeling of nonatomic events, on building state descriptions, and on the frame problem. In Section G we conclude with a brief discussion of our current work on building a planner based on this event-oriented framework.

B Motivation and Background

B.1 A Scenario

One of the primary goals behind any representational mechanism is to capture the properties of a domain in a useful and coherent way. Consider the following scenario. Three sets of friends decide to meet for dinner at an elegant restaurant. Each person must find his or her own mode of transport to the restaurant (personal car or taxi) and the first person from each party to arrive must book a reservation for his or her group. The maitre d' at the restaurant, Felix, happens to be a somewhat mercenary fellow who gives preference to parties that bribe him. In fact, everyone knows that a \$50 bribe will guarantee eventual seating. However, he does have some scruples; he will seat a party only if a table of the correct size is available and if the entire party has arrived (members of a party must be seated simultaneously). All other things being equal, he will then seat parties on a first-come-first-served basis. After being seated, guests may then choose, order, and eat their meals.

This scenario, though somewhat complex, is typical of situations we confront in our day-to-day lives. To describe it requires representation of several interlocking synchronization

constraints (all of Felix's seating rules) as well as multiple ways of achieving goals (traveling to the restaurant by car or taxi). It also manifests some naturally emerging forms of structure: individuals are grouped into parties; certain locations of activity may be viewed as resources at which only limited forms of activity may occur (the tables at the restaurant, the attention of Felix); knowledge is partitioned (some of Felix's actions and habits are known by all, whereas others may not be). These kinds of properties are found in many domains, including factory management and scheduling, robotics, and organizational coordination.

When planning for a domain such as this, it is clear that some activities are loosely coupled and can be planned separately (each person's plan for getting to the restaurant), while others must be tightly coordinated (the seating of the three parties). In addition, the expansion of some nonatomic actions will preserve a sense of locality and maintain constraints that may have been solved at a higher level of abstraction (for example, each person's menu selection plan). In other cases, however, nonatomic-event expansion will result in activity that spans across locations accessible to others, and may thus require rechecking after expansion (possible contention for the use of a limited number of taxis).

In Section D we shall illustrate our GEM-based representational approach by building a specification of this scenario. In Section G we outline a planner currently under construction that is based on GEM and that explicitly uses domain structure to guide the planning process. First, however, we take a brief look at other common forms of domain specification and the ways in which they might tackle the restaurant problem.

B.2 Other Approaches

Most traditional AI domain representations model the world as a sequence of states, and actions or events as relations between sets of states [25,78,89].¹ States descriptions are typically constructed in terms of a set of state predicates, and actions are defined in terms of preconditions and postconditions on state. This is the basic descriptive framework underlying classical planning systems such as STRIPS [25], NOAH [100], and other planners [112,114,116]. Some of these representations require that all events be totally ordered in time [25,78]. In others, events are ostensibly partially ordered, but it is still assumed that potentially interacting events occur in some total order that conforms to the partial order [100]. For instance, this premise underlies use of the STRIPS assumption [25]. While the STRIPS assumption can be used to determine the effect of individual events on world state, it cannot be used to determine the combined effect of simultaneous events. Since parallel, multiagent domains may sometimes even *require* that events occur at the same time (for example, two robots picking a heavy block up together), this is a definite limitation.²

Another disadvantage of traditional state-based frameworks is that they spread the representation of some kinds of properties across several action descriptions. For example,

¹Although they are often viewed as distinct, we shall use the terms "action" and "event" interchangeably.

²Recent work by Georgeff [32] has made progress in extending the situation-calculus framework to accommodate simultaneous events. A model-based approach is used, along with a semantic (rather than syntactic) frame rule. However, properties that involve particular relationships between events (such as simultaneity) must still be described in Georgeff's framework by using what is essentially event-based description. Thus, although his model is state based, explicit relationships between events are used.

to encode the restaurant scenario's seating rules, several state predicates would have to be maintained to encode the "synchronization state" of the reservation desk: who is waiting, in what order they arrived, who gave the biggest bribe, how large the parties are, what tables are available, etc. The preconditions and postconditions of reservation and seating-related actions would involve complex manipulation of and interactions among these predicates. Each of the seating constraints is essentially spread out among the descriptions of the actions that are involved. Changes in the constraints may therefore entail fairly complex and nonobvious changes in action descriptions. Clearly, it would be simpler if each of Felix's rules were encoded as a separate, succinct constraint on the relationship among domain actions.

More recent approaches to domain representation have been based on *state intervals*. Events (actions) and/or predicates are modeled as state sequences occurring over an interval of time, and domain properties are described in terms of interval-ordering constraints [3,12,58,80,105]. This form of representation has the benefit of allowing the state of the world *during* an event to be modeled explicitly and reasoned about. Event intervals can be temporally related in all possible ways (for example, they can be interleaved or simultaneous). Most interval-based models also explicitly utilize relationships between events (although sometimes only relationships between types or classes of events are allowed). However, events themselves are not considered the primary objects of the world model.

Unfortunately, many of these interval-based approaches do not capture the semantics of concurrent activity in a clear manner. For example, merely equating an event type with a set of state intervals (typically, the intervals which represent successful event occurrences) gives no clue as to how events are *achieved* or *composed* - e.g., what can or cannot be done by an agent.³ This hampers reasoning about interactions and relationships between events, as well as about how events may fail. Georgeff's recent paper elaborates this point [32].

³However, Allen and Koomen [5] do utilize a simple form of event decomposition similar to NOAH operators.

Some interval-based models also do not adequately capture existing or potential relationships between event instances. It is useful to be able to reason about specific causal event-pairs, or the particular events composing a nonatomic event, not just causal and composite relationships between classes of events. For example, in the restaurant scenario, it is important to know precisely which reservation events correspond to which seating events. Otherwise, Felix's seating rules could not be enforced. In addition, most interval-based representations employ a notion of causality that implies eventuality – i.e., if a class of events *A* has a causal relationship with a class of events *B*, then, if an event of type *A* occurs, an event of type *B* must too. It is therefore difficult to talk about situations in which an event has occurred but has not yet caused (and perhaps never will cause) its corresponding effect. For example, one might view a party's entering a restaurant and making a reservation as causing the party to be subsequently seated. However, such a seating need not necessarily materialize.⁴

Finally, none of these domain representations utilize event location (i.e. structural relationships between events) in any complex way.⁵ These kinds of relationships are important if we want to capture those aspects of a domain that are truly affected by locality – for example, forced sequentiality within certain regions, or boundaries of causal effect.

As we shall illustrate, an ontology based on structured, interrelated events has a distinctly different flavor from the interval-based approaches, although it shares with them many of the advantages over more traditional representations. Information about event relationships is captured explicitly. We can easily talk about particular event instances and their various temporal, causal, and simultaneity interrelationships, as well as how particular events constitute a specific nonatomic event. Event intervals and interval relationships can also be utilized. Complex structural relationships among events (i.e. various kind of event locations and groupings of locations) are also represented. The temporal logic underlying our model has a well-understood semantics and has been used extensively in concurrency theory [88,66]. Our use of temporal-logic constraints over *history sequences* (sequences of accumulating records of past behavior) is distinct from most previous approaches (although Stuart uses a similar idea [111]). Because histories include all information about previous events and their interrelationships, they facilitate use of complex information about the past.

B.3 GEM: An Informal View

Because our approach is somewhat unconventional, it is useful to begin with an informal description of the GEM world model. This discussion will be formalized later.

Our event-oriented representation is based on the view that the world can be modeled as a myriad of interrelated events occurring at locations (see Figure 5.2). The most basic

⁴While it is nonstandard, we have found it advantageous to view causality as a phenomenon more akin to *enablement*. To say that class *A* causes class *B* means that any event of type *B* must have been "enabled by" an event of type *A*. However, an occurrence of an event of type *A* does not guarantee that it will cause an event of type *B*. If, however, such a relationship *does* exist, it is perfectly reasonable to say that it is causal. If an eventuality requirement is also desired, it must be stated explicitly.

⁵Of course, many models do associate events with their performing agent. Nonetheless, this is a very limited form of event structure.

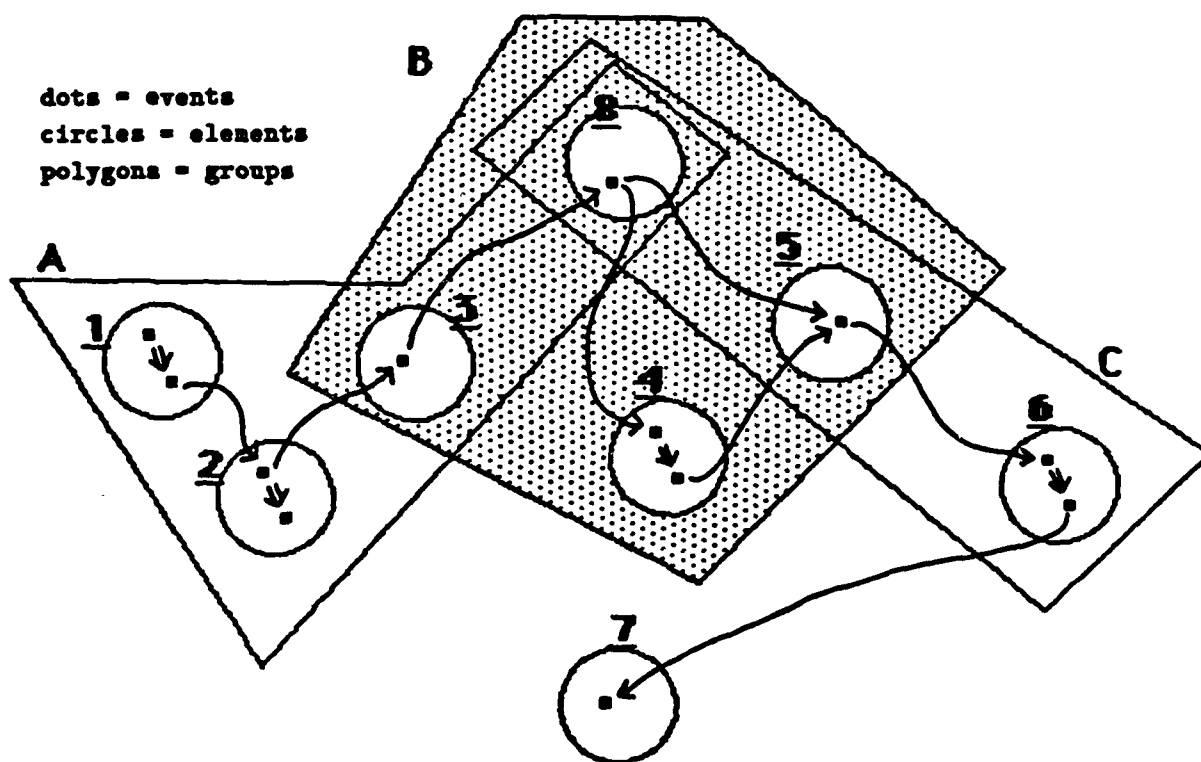


Figure 5.2: Events, Elements, and Groups

events are *atomic*; they are not observably decomposable. Nonatomic events are then composed of these atomic events. Three partial relations may hold between events: a causal relation $\sim$, a temporal order $\Rightarrow$, and a relation that embodies required simultaneity $=$. Structural relationships are also used to describe event locations as well as nonatomic event composition.

A useful way of understanding our world model is as a two-tiered structure. The upper tier is based on partially ordered sets of events related by $\sim$, $\Rightarrow$, and $=$ as well as by structural relationships. We call a particular set of interrelated events a *world plan*. (Figure 5.2 might be viewed as a pictorial representation of a particular world plan.) The lower tier of our world model consists of the set of executions *admitted by* a world plan. It is this lower tier that is usually identified as the "world model" in most state-based representations. Because each world plan may allow many possibly executions, branching state models are conventionally used to represent this execution-level view of the world. However, we have found it easier to reason about the world primarily in terms of world plans (the upper tier). These structures are definitely a more compact representation than branching state models — in fact, they correspond directly to the usual notion of a "plan."

A world plan also more clearly represents what is actually "knowable" about a domain — i.e., it models the *observable* and *necessary* qualities of a domain. For example, if $\Rightarrow (e1, e2)$ is true of a world plan, $e1$ must occur before $e2$ in every domain execution. However, if two events are observably unrelated temporally (they are unrelated by $\Rightarrow$ in the world plan), in some world executions they may occur in a particular order, while in others they may be simultaneous. (In fact, any two events that are unrelated by $\Rightarrow$ are *potentially* simultaneous.) In contrast, if $= (e1, e2)$ is true of a world plan, $e1$ and $e2$ must occur simultaneously in every world execution. The distinction between known relations and the executions admitted by them is especially useful when dealing with parallel domains. For example, people can usually perceive the known temporal order of specific world processes (and thereby can reason easily about them), but find it difficult to know exactly how these processes are interleaved (the actual world executions). World plans are thus a much more intuitive way of viewing the world than are world executions.

Because GEM's causal relation is nonstandard in some ways, it merits some additional clarification. As mentioned earlier, our causal relation $\sim$ is weaker than in other representations (for example, McDermott's [80]) in that it decouples causality from eventuality. A class of events (say *Reservations*) may hold a causal relationship to another class of events (*Seating* events), but the mere occurrence of an event of the first class does not necessarily entail that it *will inevitably* cause an event of second type. Once an event *does* cause another event, however, this relationship is represented explicitly. Of course, if an eventuality requirement is also desired, it can be specified by using a temporal logic constraint (i.e., we could say, "Every reservation must eventually cause a seating").

Our causal relation also implies an ordering in time (if $e1$ causes $e2$, it must also occur before $e2$). Therefore, we distinguish between causality and *identification* of events (for example, the identification of a light-switch-flipping event with the event of turning on the light). However, our use of the simultaneity relation $=$ enables modeling of event identification and other forms of required simultaneous activity. As in most representations

using causality, GEM's causal relation is irreducible and must be induced from outside the logic.

As mentioned earlier, events in a world plan are also clustered into locations. The most basic type of location is a locus of forced sequential activity; that is, all events belonging to such a location must be totally ordered within the temporal order $\Rightarrow$. We call these sequential locations *elements* and they are depicted in Figure 5.2 as circles.

Elements (and the events composing them) may also be grouped into larger regions of activity. We call these locations *groups*, depicted in Figure 5.2 as polygonal areas. When forms of activity logically occur at the same "location," but are not necessarily sequential, it is natural to model the location as a group consisting of many elements. For example, one might model a robot arm as a group consisting of many sequential elements. As illustrated in Figure 5.2, groups can be composed hierarchically or overlap.

Group structure is used by GEM to impose constraints on the causal relationships among events. In essence, the group boundaries may be considered just that - boundaries of causal access. Thus, groups A and C in Figure 5.2 might each be used to represent locations of activity within a robot (perhaps different segments of an arm) that can be causally related only through activity in group B (a joint). In contrast, the activity at element 7 is accessible to all locations (can be caused by activity at all locations - i.e., it is *global*). The activity at element 8 is not only accessible to activity within groups A, B, and C, but, because it is part of groups A, B, and C, it can affect or cause activity at all locations (it could perhaps represent the robot's brain). A group may also be associated with *ports* or access holes that serve as causal interfaces between the group and the events outside it.

The "state" of the world at any particular moment is modeled in GEM as the sum of past activity that has occurred. States embody not only what is true at particular moments, but also what has occurred in the past; we therefore call them *histories* or *pasts*. This view of state as an event history actually conforms quite naturally to what humans know about the world. If we allow events that model *observations*, then what can possibly be known at any particular moment (i.e. the known state of the world) will be derivable from the events that have occurred (i.e., past observations and actions). If we further categorize past activity into those sets of events occurring at, or observable from, particular sets of locations - say, those associated with the particular group modeling a robot - we can then model the "beliefs" of that robot as its localized view of past activity.

GEM's entire representational capability is built upon the basic framework we have just described. A GEM *specification* describes a particular domain by delineating the kinds of events and event structures found in the domain and then imposing constraints on the set of possible world plans (and therefore on the set of possible world executions). In the GEM specification language, constraints on actions and their interrelationships are formulated in first-order temporal logic over sequences of histories (world executions). These constraints may be scoped or applied within locations of activity. All of these capabilities are formalized and illustrated in the next two sections.

C Domain Model

As stated in the preceding section, the GEM model may be viewed as two-tiered: the upper tier models the world in terms of *world plans* (sets of interrelated events, elements, and groups); the lower tier models the actual physical executions allowed by world plans. Each world plan is composed of a set of unique objects, called *events*, that are related by a temporal ordering $\Rightarrow$, a causal relation $\sim$, and a simultaneity relation $\equiv$. Events are also grouped into *elements* which may further belong to *groups* (groups may also belong to surrounding groups).

$$W = \langle E, EL, G, \Rightarrow, \sim, \equiv, \varepsilon \rangle$$

- E = A set of event objects
- EL = A set of element objects
- G = A set of group objects
- $\Rightarrow$: $(E \times E)$ The temporal ordering
- $\sim$: $(E \times E)$ The causal relation
- $\equiv$: $(E \times E)$ The simultaneity relation
- ε : $(E \times (EL \cup G)) \cup ((EL \cup G) \times G)$ A subset relation between events and elements or groups in which they are contained, as well as between elements and groups and the surrounding groups in which they are contained.

For now we assume that all events are atomic. Thus, each event in a world plan models an atomic event that has occurred in the world domain, each relation or ordering relationship models an actual relationship between domain events, and each element or group models a logical location of activity. In Section E we shall extend this basic model to accommodate nonatomic events. Note that our assumption of event atomicity does not imply that events are totally ordered; they *may* happen simultaneously. From an intuitive standpoint, it might be useful for the reader to view each atomic event as the endpoint of some logical world action.

Every event in a world plan must be distinct; it may be viewed as a unique token. Events may be parameterized and may also be organized into types, each of which represents some class of world events. For example, *Paint(Object, Color)* could represent the class of world events, each of which paints an object a certain color. A specific instance of this type would be *paint(ladder, red)*. Lowercase tokens are used to denote specific event instances, while uppercase is used for event classes or types. A similar convention is used for parameter values and types, as well as for group and element instances and types.

As described earlier, events are related by three kinds of partial relationships, $\Rightarrow$, $\sim$, and $\equiv$. The temporal order $\Rightarrow$ is an irreflexive, antisymmetric, transitive relationship that

models event ordering in time.⁶ The causal relation $\rightsquigarrow$ is irreflexive and antisymmetric, but *not* transitive – it represents “direct” causality between events. Every domain is, by default, associated with a constraint that requires causally related events to also be temporally related ($(\rightsquigarrow(e1,e2) \supset \implies(e1,e2))$), but the reverse is not true; just because two events may be forced to occur in some sequence does not mean that they are causally related. Finally, the simultaneity relation $\equiv$ is reflexive, symmetric, and transitive, and models a *necessary* simultaneous occurrence of events.

In addition to being ordered, events are also clustered into *elements*. These elements (as well as other groups) are further clustered into *groups*. The events considered part of a group are precisely those belonging to the group’s constituent elements and groups. The structure of a domain is conveyed by the set membership relation ε .

Particular domains are represented in GEM by domain specifications. Each specification will allow a set of world plans and the world executions conforming to those world plans. A specification is built by delineating locations of activity (elements and groups), stating the types of events that may occur at these locations, and, finally, imposing constraints on the possible relationships among those events. Some of these constraints are domain-specific (for example, the constraints that describe the seating rules of the restaurant domain); others apply to all domains (e.g., the total ordering constraint on events belonging to the same element). The basic form of a constraint is a first-order temporal-logic formula that is applied to the possible executions of a given world plan. The next section describes a semantics for such constraints.

C.1 World Executions: Histories and History Sequences

A world plan, as we have described it, is actually a structure that captures or represents many potential executions in the domain being modeled. For example, consider the world plan in Figure 5.3. It can actually be executed in three ways:

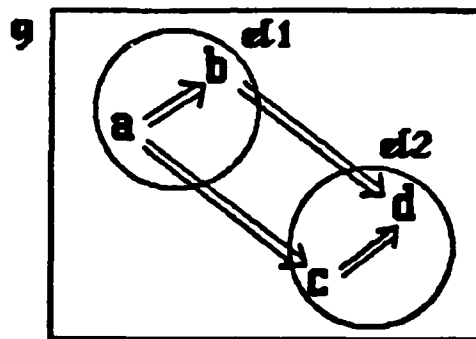
Execution 1:	1st <i>a</i>	2nd <i>b</i>	3rd <i>c</i>	4th <i>d</i>
Execution 2:	1st <i>a</i>	2nd <i>c</i>	3rd <i>b</i>	4th <i>d</i>
Execution 3:	1st <i>a</i>	2nd <i>b,c</i>	3rd <i>d</i>	

Note that, in the third execution, *b* and *c* occur simultaneously. Although we know that *one* of these world executions may occur, we cannot assume any one of them actually does.

The possible executions of a world plan may be viewed as linear sequences of “states,” where each state is a record of past activity. We call such a state a *history* or *past*.⁷ Each *history* α of a world plan W is simply a set of partially ordered events that is a *prefix* of

⁶Note that we make no use of explicit time values. In multiagent domains, it is actually disadvantageous to rely on such values for purposes of synchronization; the use of a partial temporal ordering is a much safer avenue for assessing the relative occurrences of events. However, actual time can be incorporated into GEM by associating each event with a time parameter and requiring that the temporal ordering conform to these event time stamps: $(\forall e1(t1), e2(t2)) [t1 < t2 \supset \implies (e1(t1), e2(t2))]$.

⁷The reader should be warned that the term *history* has been used by others in different ways – for example, for a particular sequence of states. Here the term refers to a snapshot of the past – i.e. a record of the events that have occurred and their interrelationships.



$$\begin{array}{cccc}
 \Rightarrow (a, b) & \Rightarrow (a, c) & \Rightarrow (b, d) & \Rightarrow (c, d) \\
 \varepsilon(a, el1) & \varepsilon(b, el1) & \varepsilon(c, el2) & \varepsilon(d, el2) \\
 \varepsilon(el1, g) & \varepsilon(el2, g) & &
 \end{array}$$

Figure 5.3: A World Plan

that world plan; it therefore may be described in the same way as a world plan, i.e., for history α , we could use $\langle E_\alpha, EL_\alpha, G_\alpha, \Rightarrow_\alpha, \sim_\alpha, \equiv_\alpha, \varepsilon_\alpha \rangle$, where $E_\alpha \subset E$, $EL_\alpha \subset EL$, $G_\alpha \subset G$, $\Rightarrow_\alpha$ is a subrelation of $\Rightarrow$, etc. Each history represents a possible point in an execution of the world plan, plus everything that has happened until then. Essentially, it is a record of past activity up to some moment in time.

For the world plan in Figure 5.3, there are six possible histories or pasts, consisting of the following sets of events (as well as their interrelationships):

$$\alpha_0 : \{\} \quad \alpha_i : \{a\} \quad \alpha_j : \{a, b\} \quad \alpha_k : \{a, c\} \quad \alpha_m : \{a, b, c\} \quad \alpha_n : \{a, b, c, d\}$$

For instance, state α_j describes the state in which events a and b have occurred, and in which, moreover, the relations $\Rightarrow_\alpha$, (a, b) , ε_α , $(a, el1)$, ε_α , $(b, el1)$, and ε_α , $(el1, g)$ all hold.

Given this notion of history ("state"), the possible world executions permitted by a world plan may be described as sequences of histories — which we shall call *valid history sequences* (VHS). For each VHS, every history in the sequence (except the first) must be a superset of its predecessor. Moreover, two events may enter a given VHS in the same history only if it is possible for them to have occurred simultaneously, i.e., if there is no explicit temporal ordering relationship between them. For example, if $\Rightarrow(e1, e2)$, then $e1$ and $e2$ would have to enter a VHS in distinct histories. By the same token, if two events *must* take place simultaneously (e.g., $\equiv(e1, e2)$), they must always enter a given VHS in the same history. A history sequence is said to be *complete* if it starts with the empty history.

For the world plan in Figure 5.3, there are three possible complete VHSs — S1, S2, and S3 — corresponding to the three possible executions of the world plan given earlier:

$$S1: \alpha_0 \quad \alpha_i \quad \alpha_j \quad \alpha_m \quad \alpha_n \equiv 1st \ a \quad 2nd \ b \quad 3rd \ c \quad 4th \ d$$

$$S2: \alpha_0 \quad \alpha_i \quad \alpha_k \quad \alpha_m \quad \alpha_n \equiv 1st \ a \quad 2nd \ c \quad 3rd \ b \quad 4th \ d$$

$$S3: \alpha_0 \quad \alpha_i \quad \alpha_m \quad \alpha_n \equiv 1st \ a \quad 2nd \ b, c \quad 3rd \ d$$

Note that one way of representing the possible history sequences of a world plan might be as a branching tree. For example, we would have

$$\begin{array}{ccccccc} & & \nearrow & \alpha_j & \rightarrow & \alpha_m & \rightarrow & \alpha_n \\ \alpha_0 & \rightarrow & \alpha_i & \rightarrow & \alpha_k & \rightarrow & \alpha_m & \rightarrow & \alpha_n \\ & & \searrow & \alpha_m & \rightarrow & \alpha_n \end{array}$$

This corresponds to the branching tree of states used by McDermott; a chronicle corresponds to a VHS [80]. However, by representing this tree as a *world plan* (i.e., the form depicted in Figure 5.3), information about possible world executions and observable relationships between events is conveyed in a more compact form.

C.2 Constraint Semantics

Now that we have valid history sequences, we have a framework for defining the semantics of formulas in first-order linear temporal logic. First, we consider simple nontemporal first-order formulas Q . Each such formula is composed in the standard fashion, with the usual quantifiers and connectives ($\wedge, \vee, \neg, \supset, \iff, \forall, \exists, \exists!$).⁸ Event, element, and group instances, as well as event-parameter values, are used as constants, over which range event, element,

⁸The quantifier $\exists!$ denotes existence of a unique object.

group, and event-parameter variables. The predicates that may be used in these formulas are *occurred*, the infix predicates $\sim$, $\Rightarrow$, $\equiv$, ε , and equality, as well as arithmetic comparison of parameter values. The interpretation of formulas is standard. Given a history α described by $\langle E_\alpha, EL_\alpha, G_\alpha, \Rightarrow_\alpha, \sim_\alpha, \equiv_\alpha, \varepsilon_\alpha \rangle$, we have

$$\begin{aligned} \alpha \models \text{occurred}(e) &\equiv e \in E_\alpha \\ \alpha \models e1 \Rightarrow e2 &\equiv \Rightarrow_\alpha(e1, e2) \\ \alpha \models e1 \sim e2 &\equiv \sim_\alpha(e1, e2) \\ \alpha \models e1 \equiv e2 &\equiv \equiv_\alpha(e1, e2) \\ \alpha \models x \varepsilon y &\equiv \varepsilon_\alpha(x, y) \end{aligned}$$

A typical nontemporal first-order formula is the following: $(\forall e:E)[\text{occurred}(e) \supset (\exists f:F)[f \sim e]]$. This might be read as follows: "Every event of type E that has occurred must have been caused by an event of type F." Free variables in a formula are considered, by default, to be universally quantified.

Linear temporal operators are modal operators that apply formulas to sequences.⁹ The most common temporal operators used are $\Box$ (*henceforth*), $\Diamond$ (*eventually*), $\bigcirc$ (*next*), and $\cup$ (*weak until*). In most temporal logics they apply formulas to sequences of states [88]. GEM follows traditional formulations of linear temporal logic, but applies the modal operators to sequences of histories (i.e., to VHSs). Given a valid history sequence of the form $S = \alpha_0, \alpha_1, \dots$, we use the notation $S[i]$ to denote the i^{th} tail sequence of S - i.e., $\alpha_i, \alpha_{i+1}, \dots$. Note that $S = S[0]$. Also notice that every tail sequence of a VHS is also a VHS.

We then define the semantics of temporal formulas as follows:

$$\begin{aligned} \text{Henceforth } P : S[i] \models \Box P &\equiv (\forall j \geq i) S[j] \models P \\ \text{Eventually } P : S[i] \models \Diamond P &\equiv (\exists j \geq i) S[j] \models P \\ \text{Next } P : S[i] \models \bigcirc P &\equiv S[i+1] \models P \\ \text{P Until Q} : S[i] \models P \cup Q &\equiv (\forall j \geq i) S[j] \models P \vee \\ &\quad (\exists j \geq i)[S[j] \models Q \wedge (\forall k, i \leq k < j) S[k] \models P] \end{aligned}$$

A nontemporal formula Q is true of $S[i]$ if it is true of α_i : $S[i] \models Q \equiv \alpha_i \models Q$.

To enhance the specification of properties dealing with past activity, we also introduce the backwards temporal operators Δ (*before*), $\bar{\Box}$ (*until now*), $P \stackrel{Q}{\leftarrow}$ (*Q back to P*), and *INIT* (*initially*). Although somewhat nonstandard, they have been used elsewhere [7].

$$\begin{aligned} S[i] \models \Delta P &\equiv S[i-1] \models P \\ S[i] \models \bar{\Box} P &\equiv (\forall k, 0 \leq k \leq i) S[k] \models P \\ S[i] \models P \stackrel{Q}{\leftarrow} &\equiv S[i] \models \bar{\Box} Q \vee \\ &\quad (\exists j, 0 \leq j \leq i)[S[j] \models P \wedge (\forall k, j < k \leq i) S[k] \models Q] \end{aligned}$$

⁹This is in contrast to branching-time temporal logics, which regard time as a branching tree. In these logics, the modalities can vary according to how they are applied to the various paths through that tree. We have found linear temporal logic to be adequate and simpler to use.

We define $INIT P$ to be $\Box [(\neg(\exists e)occurred(e)) \supset P]$. In other words, P is true of the empty history. At the beginning of a VHS, $S[0] \models \Delta P$ is *false* for all P .

Because we shall be creating structured specifications in which constraints are imposed on limited contexts, it is also useful to define *localized* or *scoped* versions of the temporal operators. For example, what happens *next* in a particular context may be different from what happens next in the domain as a whole.

Suppose we have a VHS of form $\alpha_0 \dots \alpha_n$. Let us assume that *context* is a set of events. We then define $\alpha_0 \dots \alpha_n|_{context}$ to be the history sequence remaining after histories that satisfy the following formula have been removed:

$$occurred(e) \wedge \Delta \neg occurred(e) \supset \neg e \in context.$$

In other words, we eliminate from $\alpha_0 \dots \alpha_n$ all histories that are formed solely through the addition of events outside *context*.

Given any valid history sequence S , we then define the scoped temporal operators as follows:

$$S \models \Box_{context} P \equiv S|_{context} \models \Box P$$

$$S \models \Diamond_{context} P \equiv S|_{context} \models \Diamond P$$

$$S \models O_{context} P \equiv S|_{context} \models O P$$

and similarly for other temporal operators.¹⁰

Finally, first-order temporal logic formulas may be applied to a GEM *world plan* by viewing a world plan W as the set of all its complete valid history sequences. A world plan satisfies a constraint if and only if all tail sequences of its complete valid history sequences satisfy that constraint

$$W \models P \equiv (\forall \text{ complete VHS } S \text{ of } W)(\forall i \geq 0)S[i] \models P.$$

For example, the world plan in Figure 5.3 satisfies $\Box(occurred(d) \supset b \implies d \wedge c \implies d)$, but not $\Box(occurred(c) \supset occurred(b))$ (VHS S2 does not satisfy it).¹¹

¹⁰The use of contexts may also be convenient for describing scoped or localized forms of state. For example, a state of the world α relative to a particular context would be a state α' , where all noncontextual events have been removed. If a particular context corresponds to the events that are part of, or visible to, a particular agent, then the scoped state with respect to that agent would correspond to the agent's perspective upon, or beliefs about, the past.

¹¹The temporal operator $\Box$ can actually be removed from both of these constraints, because they apply to *all* tails of complete VHSs of a world plan.

D Domain Specifications

In the next three sections, we demonstrate how GEM domain specifications are built by actually constructing a description of the restaurant scenario presented earlier. We begin with an overview of the general structure of the GEM specification language and describe how typical kinds of constraints are formed. In Sections E and F we address such issues as the specification of nonatomic actions, state description, and the frame problem. A complete specification of the restaurant scenario is given in the appendix.

D.1 Specification Structure

The GEM specification language is a set of notational conventions for writing constraints on world plans. Viewed semantically, a specification σ is equivalent to (can be expanded into) a set of first-order temporal-logic formulas over valid history sequences. Each specification defines a class of world plans by stating explicitly: (1) what types of events may occur; (2) how those events must be clustered into elements and how elements and groups are clustered into groups; and (3) what constraints exist on relationships between events and their parameter values.

Just as elements and groups model the structural aspects of a domain, they also serve as the structural components of our specification language. Each specification σ consists of a set of element and group declarations, along with a set of explicit constraints on the events that belong to those elements and groups. Each element is associated with a set of event types, and each group is composed of a set of elements and other subgroups. The events belonging to an element may be only of the designated types associated with it. The events belonging to a group are taken to be those belonging to the group's elements and subgroups. Constraints are "scoped" within the context in which they are declared; i.e., they are imposed only on those events that belong to the element or group with which they are associated. However, the temporal operators are scoped with respect to a context only if scoped temporal operators are used. Thus, if we write $\bigcirc P$, then P must be true of the next state in the *entire world execution*, not just the next state in which an event in the particular element or group occurs.

GEM also includes a mechanism for describing element and group *types*. These may be parameterized and are definable as refinements of other previously defined types. Each instance of a defined type is a unique element or group with a structure identical to that of its type description. From a semantic standpoint, the use of types and instances may be viewed as a simple text substitution facility; each type instance is shorthand for a separate but identical element or group declaration.

For example, we might describe the class of restaurant tables as follows:¹²

```

RestaurantTable (size:INTEGER) = ELEMENT TYPE .
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
  :
END RestaurantTable

```

A declaration of the form

```
table[1..5] = RestaurantTable(10) ELEMENT
```

would declare `table[1]...table[5]` to be tables of size 10. The notation `table[1].size` yields `table[1]`'s size value (in this case, 10). `table[1].Occupy` and `table[1].Vacate` refer to the class of `Occupy` and `Vacate` events belonging to `table[1]`, respectively. The notation `table[1].occupy(p)` denotes a particular `Occupy` event instance.¹³

The structure laid out by a set of group and element declarations creates a framework associated with implicit (default) constraints. Domain-specific constraints are then added on top of this framework. Default constraints include the following (we give only an informal description of these constraints here; for a more formal description, see [65]):

- The only events, elements, and groups allowed within a valid world plan are those delineated by the specification. We are essentially minimizing the potential structure of world plans with respect to the domain specification. Events must be clustered into elements and element/group structures must be formed as described in the specification.
- All events belonging to the same element must be totally ordered temporally:
 $(\forall e1, e2, elem) [e1 \in elem \wedge e2 \in elem \wedge e1 \neq e2 \supset e1 \Rightarrow e2 \vee e2 \Rightarrow e1].$
 For instance, we might represent the restaurant lobby as follows:

¹²This description models only two types of events that can take place at a table. Of course, if we wish a table to be associated with broader forms of activity, it could be modeled as an element with more event types or, alternatively, as a group consisting of many elements. For example, to represent the simultaneous lifting of both sides of a table, each side of a table could be modeled as an element associated with "lifting" events. We could also model these events as being performed by the agents that do the lifting. We could even do both (have lifting events at the table and the lifting agents) and identify the two. This form of event identification is illustrated in Section D.4.

¹³A more typical event notation might be `occupy(table[1],p)`. However, we have found dot notation to be very useful for denoting events that occur at particular elements or groups.

```

lobby = ELEMENT
EVENTS
  Enter(f:Friend)
END lobby

```

While lobby is not associated with any explicit constraint, its events must still be totally ordered – i.e., people may enter the lobby only one at a time.

- As stated earlier, we use groups as a way of representing limitations of causal effect. Essentially, the “walls” of a group form a boundary through which causal effect may probe outward, but not inward.¹⁴ The one exception to this rule is the use of *ports*: “holes” in the group boundary. If an event is a port for a group g , that event can be affected by other events outside g . Let us assume that the atomic formula $port(e, g)$ is true for every event e that is a port of group g . The formal constraints on the causal relation imposed by group structure may be described as follows.

Suppose that $e1 \in el1$ and $e2 \in el2$. Then $e1$ may cause $e2$ ($e1 \leadsto e2$) only if:

$$access(el1, el2) \vee [port(e2, g) \wedge access(el1, g)].$$

We define $access(x, y)$ to be true if either (1) x and y belong to the same group or (2) there is some surrounding group g' such that y belongs to g' and x is contained within g' – i.e., y is “global” to x . We say that an element el or group g belongs to a group g' if it is explicitly declared as one of the components of group g' . We say that el (or g) is contained within g' if there is some hierarchical scoping of groups $g1...gn$ such that el belongs to $g1$, $g1$ belongs to $g2$, ... and gn belongs to g' . (By convention, we assume that all elements and groups modeling the world are contained within a single surrounding group.)

For example, the specification structure for the restaurant domain is shown in Figure 5.4 (only one party with one friend, one taxi, and one restaurant table are depicted). Notice how the friend has access to the taxi, the reservation desk, the restaurant lobby, and also to the Vacate actions at the table (an asterisk marks port event types). However, a friend cannot directly affect Felix’s personal observations, nor can she directly occupy a table (guests must be seated by Felix).

We now define some of the domain-specific constraints for the restaurant domain. Because many constraint forms arise repeatedly, it is useful to have abbreviations for them. We shall take some liberties in devising these abbreviations, appealing to the reader’s intuition. However, all of these constraints are more rigorously defined elsewhere [65].

¹⁴This is much like the notion of scope in programming languages, except that groups may overlap as well as form hierarchies.

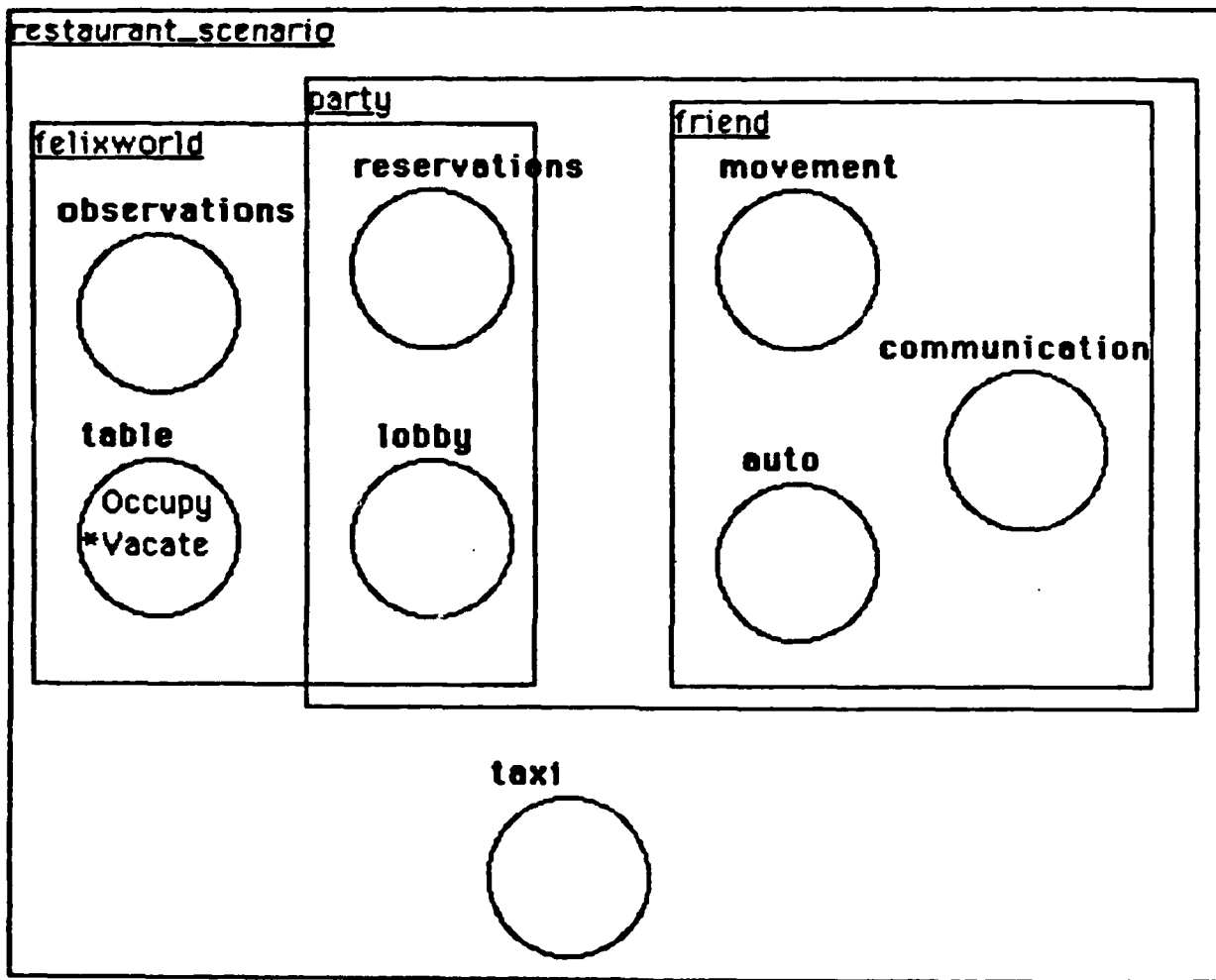


Figure 5.4: Restaurant Domain Structure

D.2 Prerequisite Constraints

Probably the most common kind of constraint is the *strong prerequisite*, denoted $E1 \longrightarrow E2$. This constraint requires that each event of type $E2$ be caused by exactly one event of type $E1$, and that each event of type $E1$ can cause at most one event of type $E2$. In essence, this is a one-to-one causal requirement. The definition of this constraint is as follows:

$$\begin{aligned} E1 \longrightarrow E2 \equiv & \\ & (\forall e2 : E2)(\exists! e1 : E1)[e1 \rightsquigarrow e2] \wedge \\ & (\forall e1 : E1)(\exists \text{ at most one } e2 : E2)[e1 \rightsquigarrow e2] \end{aligned}$$

In Section D.3 we use a strong-prerequisite constraint to describe the one-to-one causal relationship between reservation and seating events: $\text{Reserve} \longrightarrow \text{Seat}$. We can also use it to define the constraints of the `RestaurantTable` element type. Constraint 1 uses a regular-expression notation as shorthand for a more complicated pattern of strong-prerequisite constraints.¹⁵ Each `Vacate(p)` event must have a one-to-one causal relationship with a preceding `Occupy(p)` event, and each `Occupy` event (except the first) must have a one-to-one causal relationship with a preceding `Vacate` event. This restricts the events at a table to be of the form $\text{occupy}(p1) \rightsquigarrow \text{vacate}(p1) \rightsquigarrow \text{occupy}(p2) \rightsquigarrow \text{vacate}(p2) \rightsquigarrow \dots$. Namely, a table is a resource that can be used by only one party at a time.¹⁶

```
RestaurantTable (size:INTEGER) = ELEMENT TYPE
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
  1) ( Occupy(p)  $\longrightarrow$  Vacate(p) )* $\longrightarrow$ 
END RestaurantTable
```

Taxis and cars are other resources that are described in a similar fashion. In the following we use the element type hierarchy to first describe a `Vehicle` element type, and then further define taxis and personal automobiles as `Vehicles`.

¹⁵ $(x)^* \longrightarrow$ denotes zero or more repetitions of x separated by $\longrightarrow$. A formal semantics for prerequisite expressions is given in my dissertation [65].

¹⁶ Note that, although a party may "occupy" a table only once before it vacates, this does not preclude members of a party from leaving the table in the interim. As we shall see later, while some of the seating actions of party members are identified with "occupying" the table, not all are. Likewise not all departures from the table are identified with the party's actually vacating the table.

```

Vehicle = ELEMENT TYPE
EVENTS
  Occupy
  Drive(loc1,loc2:Location)
  Vacate
CONSTRAINTS
1) ( Occupy  $\longrightarrow$  Drive(loc1,loc2)  $\longrightarrow$  Vacate ) $\longrightarrow$ 
END Vehicle

```

```

Taxi = Vehicle ELEMENT TYPE
Auto = Vehicle ELEMENT TYPE

```

Another useful form of prerequisite constraint combines strong prerequisite constraints to model activity that forks or splits from an event, or joins into an event:

$$\begin{aligned}
 \text{EventFork} : E \longrightarrow \{E1, \dots, En\} &\equiv (\forall i, 1 \leq i \leq n) E \longrightarrow Ei \\
 \text{EventJoin} : \{E1, \dots, En\} \longrightarrow E &\equiv (\forall i, 1 \leq i \leq n) Ei \longrightarrow E
 \end{aligned}$$

Another common form of prerequisite relationship between events is the nondeterministic prerequisite $\{E1 \dots En\} \longrightarrow +E$, defined as follows:

$$\begin{aligned}
 \{E1, \dots, En\} \longrightarrow +E &\equiv \\
 (\forall e : E) (\exists ! ei : \{E1, \dots, En\}) [ei \rightsquigarrow e] \wedge \\
 (\forall ei : \{E1, \dots, En\}) (\exists \text{ at most one } e : E) [ei \rightsquigarrow e]
 \end{aligned}$$

In other words, each event of type E must be caused by exactly one event of type $E1$ or $E2$ or... En , and an event of type $E1$ or... En can cause at most one event of type E . This is useful for specifying situations in which exactly one (undetermined) member of a set of possible event types can cause another event.

Another form of nondeterministic behavior is the nondeterministic fork:

$$\begin{aligned}
 E \longrightarrow +\{E1, \dots, En\} &\equiv \\
 (\forall ei : \{E1, \dots, En\}) (\exists ! e : E) [e \rightsquigarrow ei] \wedge \\
 (\forall e : E) (\exists \text{ at most one } ei : \{E1, \dots, En\}) [e \rightsquigarrow ei]
 \end{aligned}$$

In other words, all events of type $E1$ or ... En must be caused by an event of type E , but each event of type E can cause only one such event.¹⁷

¹⁷ It is interesting to note that that both the nondeterministic prerequisite and the nondeterministic fork can be described using the strong prerequisite if we combine an event class set into a single event class. For example, let $F = E1 \cup \dots \cup En$, then $\{E1, \dots, En\} \longrightarrow +E \equiv F \longrightarrow E$ and $E \longrightarrow +\{E1, \dots, En\} \equiv E \longrightarrow F$.

D.3 Priority, Mutual Exclusion, Eventuality

To specify Felix's seating rules, we need a way of describing synchronization properties among events. This is easily accomplished with temporal formulas. First we define the abbreviation $e1 \text{ cbefore } E2$ ($e1$ is causally before the class of events $E2$) as follows:

$$e1 \text{ cbefore } E2 \equiv \text{occurred}(e1) \wedge \neg(\exists e2 : E2)[e1 \rightsquigarrow e2]$$

This may be read, " $e1$ has occurred but has not yet caused an event of type $E2$."

We can now express priority and mutual-exclusion properties by using the following kinds of constraints:

- *Priority of causal transitions from $e1$ to events of type $E2$ over those from $e3$ to events of type $E4$:*

$$(e1 \text{ cbefore } E2 \wedge e3 \text{ cbefore } E4) \supset \Box [(\exists e4 : E4)e3 \rightsquigarrow e4 \supset (\exists e2 : E2)e1 \rightsquigarrow e2]$$

In other words, if $e1$ is pending at $E2$ and $e3$ is pending at $E4$ at the same time, then, from that moment on, if $e3$ actually does cause an event $e4$, $e1$ must have already caused an event $e2$ ($e1$ must cause its corresponding event before $e3$ does).

- *Mutual exclusion between intervals in which $e1$ and $e3$ are causally pending:*

$$\neg(e1 \text{ cbefore } E2 \wedge e3 \text{ cbefore } E4)$$

If we define $tbefore$ as follows:

$$e1 \text{ tbefore } E2 \equiv \text{occurred}(e1) \wedge \neg(\exists e2 : E2)[e1 \Longrightarrow e2]$$

then temporal forms of mutual exclusion and priority can be expressed as well — e.g., constraints of the form

$$\neg(e1 \text{ tbefore } E2 \wedge e3 \text{ tbefore } E4).$$

Going back to the restaurant scenario, we have the following description of Felix's reservation desk:

reservations = ELEMENT

EVENTS

Reserve(p:Party, b:Bribe)

Seat(p:Party, t:RestaurantTable)

CONSTRAINTS

1) To be seated, a party must have a reservation. Moreover, each reservation is good for only one seating.

$\text{Reserve}(p,b) \longrightarrow \text{Seat}(p,t)$

2) Parties can be seated only at tables of the right size.

$\text{occur}(\text{seat}) \supset \text{seat.p.size} = \text{seat.t.size}$

3) A bigger bribe will get you seated faster.

$\text{reserve1} \text{ cbefore } \text{Seat} \wedge \text{reserve2} \text{ cbefore } \text{Seat} \wedge \text{reserve1.b} > \text{reserve2.b} \supset$

$\Box [(\exists \text{seat2:Seat}) \text{reserve2} \leadsto \text{seat2} \supset (\exists \text{seat1:Seat}) \text{reserve1} \leadsto \text{seat1}]$

4) All other things being equal, seating is first-come-first-served.

$\text{reserve1}(p1,b1) \implies \text{reserve2}(p2,b2) \wedge b1=b2 \wedge$

$\text{present}(p1) \wedge \text{present}(p2) \supset$

$\Box [(\exists \text{seat2:Seat}) \text{reserve2} \leadsto \text{seat2} \supset (\exists \text{seat1:Seat}) \text{reserve1} \leadsto \text{seat1}]$

5) A \$50 bribe will definitely get you seated.

$\text{reserve.b} \geq \$50 \supset \Diamond (\exists \text{seat:Seat}) \text{reserve} \leadsto \text{seat}$

END reservations

Note how the temporal operator $\Diamond$ (eventually) is used to describe the rule for eventual seating, given a \$50 bribe. The state description $\text{present}(p)$ represents states in which all members of party p are present in the lobby. It is defined to be true precisely at the time all the friends in party p have arrived in the restaurant lobby but have not yet been seated. The definition given below thus illustrates the use of event-based formulas to define state predicates (see Section F).

$$\text{present}(p) \equiv (\forall f:\text{Friend}, f \in p) (\exists \text{enter}(f):\text{Lobby.Enter})$$

$$\text{enter}(f) \text{ cbefore } \text{Reservations.Seat}$$

D.4 Simultaneity

One way of establishing relationships between the private events of restaurant guests and those of the restaurant's logical components is to form identifications between them. For example, we can identify a reservation event performed by one of the friends in a party with the reservation event at the restaurant desk. Similarly, we require that Felix's seating of a party coincides with a sitting action by each member of the party. These identifications will force all restaurant guests to comply with Felix's seating rules. Event identification (as well as other forms of required simultaneity) is accomplished by using the simultaneity relation $\equiv$. In particular, we use the following kinds of constraints:

$E1 \sim E2 \equiv (\forall e1 : E1)(\exists e2 : E2)[e1 = e2]$

$E1 \approx E2 \equiv (\forall e1 : E1)(\exists e2 : E2)[e1 = e2] \wedge (\forall e2 : E2)(\exists e1 : E1)[e1 = e2]$

Note that $E1 \approx E2$ is equivalent to $E1 \sim E2 \wedge E2 \sim E1$. The constraint $E1 \sim E2$ identifies all events of type $E1$ with events of type $E2$, but not vice versa. For example, all of Felix's Seat events must be identified with Sit actions by members of a party, but not all Sit actions by a particular person need be identified with seating by Felix.

We begin with a preliminary description of a friend. Each friend is made up of movement events, communication events, and use of a personal automobile. Note that events at each of the elements composing a friend must be sequential, but events occurring at different elements may be simultaneous (as long as they conform to the domain constraints). Thus, people can potentially communicate and move at the same time.

```
Friend = GROUP TYPE (m:Movement, c:Communication, a:Auto)
```

```
Movement = ELEMENT TYPE
```

```
EVENTS
```

```
  Ride(loc1,loc2:Location)
```

```
  Walk(loc1,loc2:Location)
```

```
  Sit(tableloc:Location)
```

```
  Eat
```

```
END Movement
```

```
Communication = ELEMENT TYPE
```

```
EVENTS
```

```
  Reserve(p:Party, b:Bribe)
```

```
  OrderFood(food:Food)
```

```
END Communication
```

A party consists of a set of friends, the reservation desk, and the lobby: ¹⁸

¹⁸The notation $f.c.\text{Reserve}$ denotes events of type Reserve occurring at the Communication element c of Friend f . SELF is used to denote the group constant associated with each particular group instance. Thus, when the Party type definition is instantiated, SELF will be replaced by each Party instance's group constant. The function `setsize` yields the cardinality of a set.

Party(size:INTEGER) = GROUP TYPE ({f}:SET OF Friend,reservations,lobby)

CONSTRAINTS

- 1) size must be the size of the set of friends.
 $size = \text{setsize}(\{f\})$
- 2) A reservation by a friend is identified with a reservation at the desk.
 $f.c.\text{Reserve}(p,b) \approx \text{reservations.Reserve}(p,b)$
- 3) All members of a party must be seated simultaneously.
 $(\forall f' \in \{f\}) \text{reservations.Seat}(\text{SELF},t) \sim f'.m.\text{Sit}(t)$
- 4) In order to be seated, all members of the party must be present.
 $(\forall f' \in \{f\}) \text{lobby.Enter}(f') \longrightarrow \text{reservations.Seat}(\text{SELF},\text{Table})$
- 5) The first friend to enter the lobby must make a reservation.
 $(\forall f1,f2 \in \{f\}) \text{occurred}(\text{lobby.enter1}(f1)) \wedge$
 $\neg(\exists \text{lobby.enter2}(f2)) [\text{lobby.enter2}(f2) \implies \text{lobby.enter1}(f1)] \supset$
 $\diamond (\exists \text{reserve1}:f1.c.\text{Reserve}(\text{SELF},b)) \text{occurred}(\text{reserve1})$

END Party

We can also now specify Felix's world, consisting of the reservation desk, the lobby, a set of tables `table[1]...table[10]` (these are assumed to have been instantiated), as well as Felix's personal observations or thoughts. One sort of observation that Felix may make is whether a table is empty. Later on we shall add an extra constraint that allows Felix to make such observations only if the table is indeed empty. For now, however, we assume that Felix always makes accurate observations. He will not seat a party unless *he* thinks a table is unoccupied. Note that this is different from saying that a seating may take place when the table is free.¹⁹

observations = ELEMENT

EVENTS

EmptyTable(t)

END observations

felixworld = GROUP (reservations, table[1..10], lobby, observations)

PORTS(table[i].Vacate)

CONSTRAINTS

- 1) Felix must observe that a table is empty before he can seat a party there.
 $\text{observations.EmptyTable}(\text{table}[i]) \longrightarrow \text{reservations.Seat}(\text{Party},\text{table}[i])$
- 2) Seating a party at a table is the same as occupying the table.
 $\text{reservations.Seat}(p,\text{table}[i]) \approx \text{table}[i].\text{Occupy}(p)$

END felixworld

¹⁹We could have chosen to do this as well; we just wanted to illustrate the use of "observation" events.

E Nonatomic Events and Hierarchical Description

We now digress from our development of the restaurant domain specification to discuss the use of nonatomic events within the GEM framework. The inclusion of such events is relatively straightforward once it is realized that a nonatomic event can be described by using two or more internal events. In particular, we associate each nonatomic event type E with two atomic event types E' and E'' , representing the initiation and termination of E . We also add an additional constraint: $E' \rightarrow E''$ (there must be a one-to-one causal relationship between the initial event and terminal event for each nonatomic event). A nonatomic event e is *in progress* if the formula $e' \text{ cbefore } E''$ is true. Using this notation, we can describe the various possible ordering relationships between two nonatomic events a and b as follows (these are the same interval relationships used by Allen [3]):²⁰

a before b	$\equiv a'' \Rightarrow b'$
a equal b	$\equiv a' \Rightarrow b' \wedge a'' \Rightarrow b''$
a meets b	$\equiv \text{occursnext}(a'') \supset \bigcirc \text{occursnext}(b')$
a overlaps b	$\equiv a' \Rightarrow b' \wedge a'' \Rightarrow b''$
a during b	$\equiv b' \Rightarrow a' \wedge a'' \Rightarrow b''$
a starts b	$\equiv a' \Rightarrow b' \wedge a'' \Rightarrow b''$
a finishes b	$\equiv b' \Rightarrow a' \wedge b'' \Rightarrow a''$

The use of initial and terminal events will be the basis for our addition of nonatomic events to the GEM domain model. We extend world plans to include nonatomic events, and also add a relation κ , which models the composition of a nonatomic event – i.e., if $\kappa(e, f)$, then e is a part of nonatomic event f . Thus, we now have world plans of the form

$$W = \langle E, EL, G, \Rightarrow, \sim, =, \varepsilon, F, \kappa \rangle.$$

where F is a set of nonatomic events, and $\kappa : ((E \cup F) \times F)$ is the part-of relation between atomic (or nonatomic) events and nonatomic events. We require that, in all world plans, there should exist for each nonatomic event f two atomic events f' and f'' (its initial and terminal events) such that $\kappa(f', f)$ and $\kappa(f'', f)$. We also have the following additional constraints:

$$\begin{aligned}
 &\text{occurred}(f) \supset f' \sim f'' \\
 &e \sim f \supset e \sim f' \\
 &e \Rightarrow f \supset e \Rightarrow f' \\
 &f \sim e \supset f'' \sim e \\
 &f \Rightarrow e \supset f'' \Rightarrow e \\
 &e = f \supset e \varepsilon F \wedge f' = e' \wedge f'' = e'' \\
 &f \varepsilon e \supset f' \varepsilon e \wedge f'' \varepsilon e \\
 &f \varepsilon g \supset f' \varepsilon g \wedge f'' \varepsilon g
 \end{aligned}$$

²⁰The abbreviation $\text{occursnext}(e)$ is defined by $\neg \text{occurred}(e) \wedge \bigcirc \text{occurred}(e)$.

Note that a nonatomic event can be simultaneous only with another nonatomic event. We consider them to be simultaneous if their endpoints are simultaneous. This is equivalent to Allen's relation *equal* (see [3]).

To model a nonatomic domain action, we can now simply use two atomic events – its initial and terminal events. Usually, however, it is preferable to associate nonatomic actions with a particular form of behavior. For example, we might want to associate a nonatomic event type with particular intervals over which some formula holds.

Suppose we have a formula P that is true of particular histories.²¹ By using the following constraint, we can identify a nonatomic event type F with every convex interval in which P is true:

$$\begin{aligned} &P \wedge \Delta \neg P \supset \\ &(\exists f':F') \text{ justoccurred}(f') \wedge P \cup (\neg P \wedge (\exists f'':F'') [\text{lastoccurred}(f'') \wedge f' \sim f'']) \\ &\text{where} \\ &\text{justoccurred}(f') \equiv \text{occurred}(f') \wedge \Delta \neg \text{occurred}(f') \\ &\text{lastoccurred}(f'') \equiv \Delta \text{justoccurred}(f'') \end{aligned}$$

Namely, in any history in which P becomes true, there occurs some event f' and, when P is about to become false again, f'' occurs.

Alternatively, we can choose to model nonatomic actions as particular patterns of behavior. This is actually much more appealing in an event-based framework. To describe how a nonatomic event is achieved, we use an abbreviation of the following form:

$$F \doteq E1 \longrightarrow \dots \text{event pattern} \dots \longrightarrow En$$

This states that the nonatomic event type F is composed of a pattern of other event types, beginning with an atomic *initial*-event type (here $E1$) and ending with an atomic *terminal*-event type (En). These initial and terminal event types are then identified (by using $\sim$) with F' and F'' respectively. If more than one way of achieving F is supplied, events f' and f'' for each nonatomic event f may be identified with any of the possible initial-event/terminal-event pairs of event patterns that could compose f . Finally, we require that for all events ei of type $E1 \dots En$ that compose an event f , $\kappa(ei, f)$ must hold. Any arbitrary event pattern or set of constraints may be used to describe the set of events composing a nonatomic event. Such nonatomic events are therefore very similar to the notion of process used by Georgeff and me [34].

We now return to the restaurant scenario. We can use a nonatomic-event description to define the different methods the friends have for traveling to the restaurant. Earlier we associated each friend with a *Movement* element containing an event type *Ride*(loc1,loc2). We can now view *Ride* as a nonatomic event that can be expanded in two ways:

$$\begin{aligned} \text{Ride}(\text{loc1}, \text{loc2}) &\doteq \text{auto.Occupy} \longrightarrow \text{auto.Drive}(\text{loc1}, \text{loc2}) \longrightarrow \text{auto.Vacate} \\ \text{Ride}(\text{loc1}, \text{loc2}) &\doteq \text{taxi.Occupy} \longrightarrow \text{taxi.Drive}(\text{loc1}, \text{loc2}) \longrightarrow \text{taxi.Vacate} \end{aligned}$$

²¹In other words, P is a *state description*.

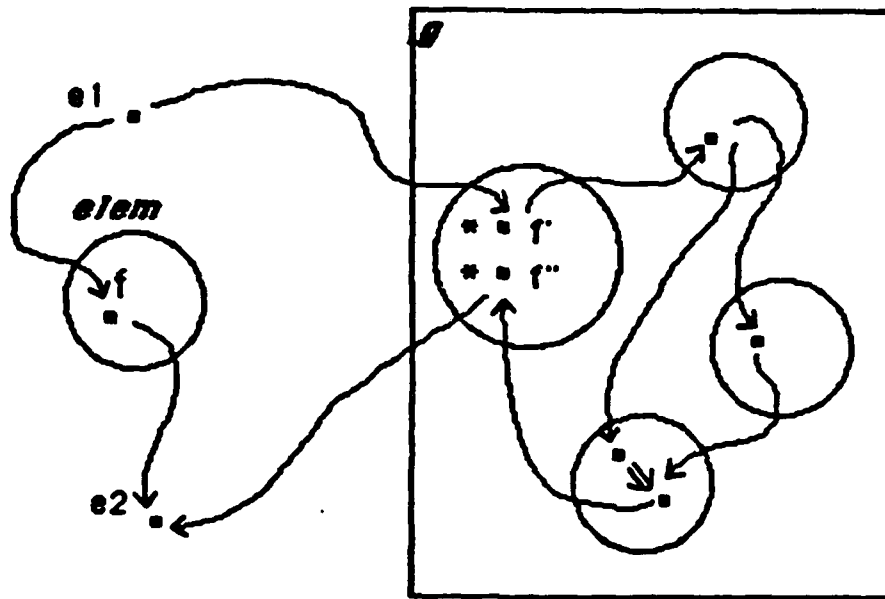


Figure 5.5: Protected Nonatomic-Event Expansion

The element variables *auto* and *taxi* must be bound to the friend's personal automobile or to any taxi, respectively.

Note that, once Ride events have been expanded, they may cause interference with other events in the domain that were not observable beforehand. For example, if there is only one taxi in town and no friend wishes to use his or her personal auto, the Ride events for all friends will, after expansion, be forced into a total ordering.

Unfortunately, it seems inevitable that the expansion of *arbitrarily* defined nonatomic events will result in added complications (or even a violation of domain constraints) at the resulting lower level of description. It is precisely for these reasons that hierarchical planners such as NOAH must *recheck* domain constraints after each event expansion. This is clearly an undesirable state of affairs. It can lead to a combinatorial explosion in the cost of reasoning about and planning in such domains.

One way of getting around this problem is to limit the forms of behavior that can constitute a nonatomic event. This limited form of behavior must lack interaction with other events in the domain; it must somehow be encapsulated. Group structure, with its associated causal limitations, is a candidate for achieving this needed encapsulation.

Consider a nonatomic event type F occurring at element *elem*. Rather than assume that initial- and terminal-event types F' and F'' also belong to *elem* (as is normally done), we create a group g with port event types F' and F'' (see Figure 5.5). Port events of the form f' and f'' serve as an interface between the rest of the domain and the protected forms of activity within group g (which compose events of type F). We use the abbreviation

$$F \doteq g \ E_1 \longrightarrow \dots \longrightarrow E_n$$

for this kind of protected or limited event expansion. All events composing an event of type F must belong to group g .

We can use protected event expansion to describe the nonatomic event type $\text{OrderFood}(\text{food})$ in the Friend specification. To each Friend group we add a new subgroup of type FoodOrdering .

$\text{FoodOrdering} = \text{GROUP TYPE } (r:\text{Sight}, t:\text{Thought}, s:\text{Speech})$

Each Sight element is assumed to have an event type $\text{Read}(\text{menu})$, Thought has an event type $\text{Choose}(\text{food})$, while Speech has an event type $\text{Utter}(\text{food})$. We then add to the Friend specification the following constraint (fo denotes the FoodOrdering group belonging to the particular Friend):

$$\begin{aligned} \text{OrderFood}(\text{food}) &\doteq fo \\ fo.\text{Sight}.\text{Read}(\text{menu}) &\longrightarrow fo.\text{Thought}.\text{Choose}(\text{food}) \longrightarrow fo.\text{Speech}.\text{Utter}(\text{food}) \end{aligned}$$

Each FoodOrdering group has no ports other than those of type $\text{OrderFood}'$ and $\text{OrderFood}''$. Thus, if no constraints refer to events in the FoodOrdering group other than those constraints explicitly associated with FoodOrdering , and if these constraints are also localized (i.e. they utilize scoped modal operators), then interactions between a friend's food ordering activity and other domain events cannot take place.

F State Description

As we have been trying to illustrate throughout this paper, many domain properties can easily be described without the use of state predicates and, in some cases, more naturally. However, such predicates are often useful for encoding or abbreviating aspects of past behavior. Thus, we might want to write a constraint of the form " P is a precondition of event e ," without also stating in that constraint how P was achieved. For example, given some sort of definition for P , we could use the Precondition constraint defined below:²²

$$\begin{aligned} \text{Precondition}(e, P) &\equiv \text{occursnext}(e) \supset P, \\ \text{where} \\ \text{occursnext}(e) &\equiv \neg \text{occurred}(e) \wedge \bigcirc \text{occurred}(e) \end{aligned}$$

The approach we will take to atomic state formulas once again emphasizes the duality between state-based and event-based representations; just as many state-based descriptions represent events as relations between states, so shall we represent state predicates dually as formulas pertaining to events. The truth or falsity of an atomic state formula will thus depend on the definition of its predicate.

In some cases, these defining formulas will form a complete description of the state predicate. For example, we might define a predicate `TableEmpty` as follows:

$$\begin{aligned} \text{TableEmpty}(\text{table}) &\equiv \\ &\neg (\exists \text{occupy}:\text{table}.\text{Occupy}) \text{ occurred}(\text{occupy}) \vee \\ &(\exists \text{vacate}:\text{table}.\text{Vacate}) \text{ vacate cbefore table.Occupy} \end{aligned}$$

If, for some history and `table`, there has never been a `table.Occupy` event or there is a `table.Vacate` event that has not yet been followed by a corresponding `Occupy` event (i.e., the table has been vacated but not yet reoccupied), then `TableEmpty(table)` is true in that history. If this formula evaluates to false, `TableEmpty(table)` is also false.

We shall call such formulas *complete predicate definitions*. Given the `TableEmpty` definition, we have the following constraint on Felix's observations:

$$(\forall \text{empty}(t):\text{observations}.\text{EmptyTable}) \text{ Precondition}(\text{empty}(t), \text{TableEmpty}(t)).$$

Notice that no frame problem arises when complete predicate definitions are used; the atomic formula `TableEmpty(table)` is defined to be true for a particular `table` if and only if its corresponding formula is true. Consequently, there can be no question about the effects of unrelated events on its truth value; once true (or false), `TableEmpty(table)` remains true (or false) for a particular value of `table` until its defining formula becomes false (or true).

Sometimes, however, we shall want to use weaker, *incomplete predicate definitions* – e.g., assertions of the form $\text{formula1} \supset P$ and $\text{formula2} \supset \neg P$. In this case, we know that, if formula1 is true, so is P . However, if formula1 is false, we *cannot* conclude $\neg P$. This

²²To say that P is a precondition of *all* events of type E , we would write: $(\forall e:E) \text{ Precondition}(e, P)$.

would be the case, however, if formula2 were true.²³ Other types of incomplete predicate definitions might include use of the temporal operators. These describe ways of attaining P for particular histories. For example, we might have $\text{formula1} \supset \bigcirc P$ or $\text{INIT } P$.

We may also want to build predicate definitions not only with behavioral (i.e., event-based) formulas, but also with formulas that utilize other state predicates. For example, we might have

$$\begin{aligned} &\neg P \vee Q \vee \text{formula1} \supset R \\ &\text{formula2} \equiv P \\ &R \vee \text{formula3} \supset Q, \end{aligned}$$

where formula1, formula2, and formula3 are purely event-based. To find valuations for P , Q , and R in a particular history, we could use an iterative-evaluation mechanism. Behavioral portions of formulas would be evaluated first and then the formulas would be iteratively simplified wherever possible. Of course, sometimes this will not yield a truth value for all atomic formulas. For example, if formula2 and formula3 are true and formula1 is false, then we have $R \wedge P \wedge Q$. However, if formula1 and formula3 are false and formula2 is true, we can conclude P but nothing more about R and Q (except that $R \iff Q$).

In order to apply incompletely-defined predicates to *all* histories in an execution, it is necessary to have some sort of frame rule to assert the persistence of P (or $\neg P$) despite the occurrence of other events. Such a rule is defined in Section F.2, which essentially minimizes the effect of events on state formulas. In the future we hope to explore the use of circumscription in the GEM framework [76].

Note that, unlike formalisms based on STRIPS-like action descriptions, the use of predicate definitions (both complete and incomplete) suffers from none of the problems created by the possibility of simultaneous action. Since effects upon a world state are not prescribed in the context of individual event descriptions (as is normally done in many state-based frameworks), there is no confusion regarding the effect of simultaneous activity or any other form of behavior. If the formula defining the truth-value of an atomic formula P is true of a history, P itself is true of the history.

For example, suppose we have a domain with two events $e1$ and $e2$, plus the requirement that, if $e1$ and $e1$ occur simultaneously, Q will be true. Otherwise P will be true. In GEM we would simply state that $e1 = e2 \supset Q$ and $\neg e1 = e2 \supset P$. These effects on P and Q cannot be expressed in the classical STRIPS framework nor in any framework that does not accommodate event simultaneity and explicit relationships between events.

F.1 Add/Delete Axioms

The add/delete lists used in STRIPS-like frameworks to define the effects of events on state can easily be cast in terms of predicate definitions. Let $\tilde{v}$, $\tilde{w}$, $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ denote tuples

²³Of course, for these definitions of P and $\neg P$ to be consistent, $\neg(\text{formula1} \wedge \text{formula2})$ must hold.

of free variables and/or constants. Event type $E(\tilde{x})$ denotes the class of events of type E whose parameters match $\tilde{x}$; and $P(\tilde{z})$ matches those atomic formulas formed from predicate symbol P and a tuple of variables or constants matching $\tilde{z}$.

Given this notation, for every event type $E(\tilde{x})$ that adds $R(\tilde{y})$ under precondition $P(\tilde{z})$, we use a declaration of the form $Adder(E(\tilde{x}), P(\tilde{z}), R(\tilde{y}))$ and, for every event type $F(\tilde{w})$ that deletes $R(\tilde{y})$ under precondition $Q(\tilde{v})$, we have $Deleter(F(\tilde{w}), Q(\tilde{v}), R(\tilde{y}))$. If these *Adder* and *Deleter* declarations characterize the effects on $R(\tilde{y})$ completely, we can use the following predicate definition for $R(\tilde{y})$:²⁴

$$R(\tilde{y}) \equiv \\ (\exists e: E(\tilde{x})) [Adder(E(\tilde{x}), P(\tilde{z}), R(\tilde{y})) \wedge occurred(e) \wedge \Box precondition(e, P(\tilde{z})) \wedge \\ \neg (\exists f: F(\tilde{w})) [Deleter(F(\tilde{w}), Q(\tilde{v}), R(\tilde{y})) \wedge \Box precondition(f, Q(\tilde{v})) \wedge e \implies f]]$$

This states that $R(\tilde{y})$ is true if and only if it has been made true and has not been subsequently deleted. Of course, we might want to create other rules using *Adder* and *Deleter* (for example, stating that something is true *unless* deleted), or use *Adder* and *Deleter* to build incomplete rather than complete definitions. For example, if we stated that the above formula only *implies* $R(\tilde{y})$, it would be equivalent to the STRIPS rule (i.e. the value of $R(\tilde{y})$ would become *undefined* after a *Deleter* event occurs). Notice that we have also assumed that all relevant *Deleter* and *Adder* events have been explicitly stated. Thus, we have invoked a form of closed-world assumption. Another alternative might have been to use some form of circumscription over *Adder* and *Deleter* specifications.²⁵ While we tend to assume some form of minimization of the effects of events in this paper, we do not wish to take a particular stand on how it is done. We intend to explore this further in future work.

F.2 The Frame Problem

In many ways, several of the difficulties often associated with the frame problem find relief in our structured event-based model. GEM's ability to structure events into elements and groups automatically imposes constraints that limit their effects upon each other. For example, events occurring within nonintersecting groups cannot causally affect one another except through explicitly-defined ports. Likewise, events occurring within the same element cannot occur simultaneously, thereby limiting the amount of reasoning necessary to determine the effects of simultaneity. Another important aspect of group/element structure is

²⁴ Note that this formula can be expanded into a first-order formula by taking a disjunction over all possible combinations of *Adders* and *Deleters*.

²⁵ Our use of the closed-world assumption here has been made only with respect to the use of *Adders* and *Deleters* for defining state predicates. It need not necessarily apply to the definition of state predicates in general. For instance, the frame rule given in the next section assumes that some way of determining which events affect which predicates is given, but does not state exactly how. Nor does the rule determine the value of an atomic formula if it is affected by an event in some *unknown* way.

that it provides a well-defined *framework* in which nonmonotonic reasoning can take place. For example, while we may discover new qualifications on previously known preconditions for starting a car (e.g., that there be no potato in the tailpipe), groups provide a structure in which to add and use these qualifications. For example, if we model the car as a group, we could add new ports to that group which allow for the newly discovered kinds of effects.

While we may effectively use group/element structure to alleviate aspects of the frame problem, there is still a need for frame rules in GEM. In particular, we still need a way to complete incompletely-defined predicates. As we have so far described, the truth or falsity of a state-based formula with respect to a particular history must be derived from predicate definitions. Given that we want to build a computational system based on our model, it will be inefficient to reevaluate these definitions for every history. This problem is somewhat alleviated by the fact that event-based domain descriptions do not often employ state predicates. Still, some sort of frame rule for carrying over atomic-formula valuations from one history to the next seems to be necessary. The same rule can also be used for extending incomplete predicate definitions to form complete predicate definitions.

We now describe a semantic frame rule of the form used by Georgeff [32]. Such semantic rules avoid the difficulties usually associated with syntactic approaches to the frame problem. For each n -ary predicate symbol P and event e in a world plan, we add a formula $\delta_P(e, \tilde{x})$ to the world plan, where $\tilde{x}$ is an n -tuple of free variables. This states that, for every $\tilde{x}$, if $\delta_P(e, \tilde{x})$ holds, then $P(\tilde{x})$ may be affected by e . (Note the similarity between δ and the *Adder/Deleter* classifications used in the previous section.²⁶) We then have the following frame rule:

$$\Delta P(\tilde{x}) \wedge \neg(\exists e)[justoccurred(e) \wedge \delta_P(e, \tilde{x})] \supset P(\tilde{x}).$$

In other words, if $P(\tilde{x})$ is true in a given history and no event occurs that can affect $P(\tilde{x})$, then $P(\tilde{x})$ remains true.

While the formula $\delta_P(e, \tilde{x})$ may certainly be different for every P , e , and $\tilde{x}$, a useful way of defining δ formulas in general is to assume that the predicate definitions in a particular specification are complete. In other words, we could assume that only events of types explicitly designated as affecting $P(\tilde{x})$ can affect $P(\tilde{x})$. Suppose we define $eventset(P(\tilde{x}))$ to be precisely the set of events so designated. We then assert: $\delta_P(e, \tilde{x}) \iff e \in eventset(P(\tilde{x}))$. The frame rule then reduces to: $\Delta P(\tilde{x}) \wedge \neg(\exists e \in eventset(P(\tilde{x})))justoccurred(e) \supset P(\tilde{x})$.

G Conclusion

This paper has presented a structured, event-based framework for representing the properties of domains with parallel activity. We have attempted to demonstrate its utility in describing the complex properties of such domains in a coherent and semantically sound

²⁶ And, as stated in the previous section, while we may wish to minimize δ , we take no stand in this paper on exactly how this is to be done.

fashion. Our model, designed explicitly for describing parallel domains, has a well understood semantics (the semantics of first-order temporal logic) that has been used elsewhere in concurrency theory. By the use of first-order temporal-logic constraints on *history sequences*, complex synchronization properties based on causality, temporal ordering, and simultaneity can be expressed easily and naturally.

An important aspect of our model is its explicit representation of event location. This is used to embody the structural aspects of a domain, to localize domain constraints, and to impose constraints on the temporal ordering as well as on causal access. The model also includes the notion of nonatomic events. State-based specifications can be incorporated by describing state in terms of past behavior. We have presented a semantic frame rule for such uses of state. However, we have also stressed the fact that many properties can be expressed without resorting to state-based description.

We are currently constructing a planning system (GEMPLAN) based on the formalism described in this paper. It is being written in Prolog on a Sun 3/50. The present system is capable of generating multiagent solutions to blocks world problems [62]. Given an event-based specification for a domain, the planner builds an initial world plan that reflects a particular problem's initial and goal states. This event network is then filled in through a process of incremental constraint satisfaction.

The planning search space in GEMPLAN may be viewed as a tree with a partial plan at each node. When a node is reached, the system checks to see whether a particular constraint has been satisfied. If it has not, the search space branches for each of the possible plan "fixes" that will satisfy that constraint. Since the kinds of constraints that must be satisfied are much broader than the pre- and postconditions used by other planners, the derivation of constraint fixes is a nontrivial problem.

Highlights of the current system include a table-driven search mechanism that can be adapted to specific domains, an efficient representation of world plans, facilities for plan explanation, and nonatomic-event expansion. Initial work has also begun on delayed binding of event parameters, on accumulating constraints on unbound variables, and on dependency-directed search and backtracking.

Especially promising is current work on structuring and guiding the search in a way that makes use of the domain structure itself. The planning space is partitioned to reflect the element/group structure of the domain and search through this partitioned space is guided by the aforementioned table-driven mechanism, which is tailorable to the domain. The result is a planning architecture that can generate plans in a manner that can vary with the specific application, ranging from loosely coordinated, distributed formalisms (e.g., less tightly synchronized applications) to more tightly coordinated formalisms (e.g., those applications in which synchronization constraints are strong and complex).

Another useful result is an algorithm that transforms an n -robot solution to an m -robot solution where $n > m$. By using this algorithm, many plans can be generated and then made less parallel as resources are reduced, without loss of generality. This development to be applicable to many other resource-constrained domains in the blocks world and we shall be reporting on it at a later date.

NO-A182 724. ONR (OFFICE OF NAVAL RESEARCH) RESEARCH IN DISTRIBUTED 2/2
REASONING AND PLANNING(U) SRI INTERNATIONAL MENLO PARK
CA ARTIFICIAL INTELLIGENCE CENTE. K G KONOLIGE ET AL.
UNCLASSIFIED MAY 87 N00014-85-C-0251 F/G 12/7 NL

ONR (OFFICE OF NAVAL RESEARCH) RESEARCH IN DISTRIBUTED
REASONING AND PLANNING(U) SRI INTERNATIONAL MENLO PARK
CA ARTIFICIAL INTELLIGENCE CENTE.. K G KONOLIGE ET AL.
MAY 87 N00014-85-C-0251 F/G 12/7

2/2

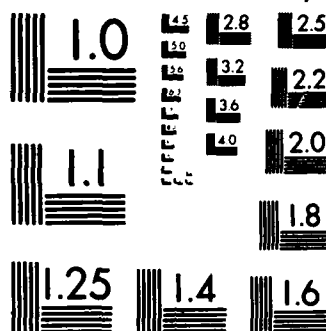
UNCLASSIFIED

MAY 87 N00014-85-C-0251

F/G 12/7

ML

ENL
8-87
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

To satisfy violated constraints, we are currently using predefined fixes for common constraint forms. Because of the intractability of solving arbitrary first-order temporal-logic constraints, we considered this to be a good initial approach to this problem. It is similar to Chapman's idea of *cognitive cliches* – i.e., utilizing a set of specialized theories that are common to many domains, rather than trying to solve for the most general theory [11]. A similar idea is incorporated in the ISIS system [26]. However, we are also working on techniques for deriving some fixes automatically from the logical form of a constraint, at least for a subset of the logic. The synchronization techniques for solving propositional temporal-logic constraints as conceived by Manna and Wolper [74] (and implemented by Stuart [111]) may eventually be applied.

Acknowledgments

I would like to thank Michael Georgeff for his critical reading of several drafts of this paper as well as many enlightening discussions on multiagent-domain representation and planning. Thanks also to David Fogelson for his help in constructing GEMPLAN and for his constructive comments in reviewing this paper.

Example: Restaurant Domain Specification

Restaurant Tables (of sizes 1 to 10)

```

RestaurantTable (size:INTEGER) = ELEMENT TYPE
EVENTS
  Occupy(p:Party)
  Vacate(p:Party)
CONSTRAINTS
  1) Tables can be occupied by only one party at a time.
    ( Occupy(p)  $\longrightarrow$  Vacate(p) ) $\ast\longrightarrow$ 
END RestaurantTable
table[i=1..10] = RestaurantTable(i) ELEMENT

```

Vehicles

```

Vehicle = ELEMENT TYPE
EVENTS
  Occupy
  Drive(loc1,loc2:Location)
  Vacate
CONSTRAINTS

```

1) Vehicles can be occupied by only one passenger at a time.
 (Occupy → Drive(loc1,loc2) → Vacate)*→
END Vehicle

Taxi = Vehicle ELEMENT TYPE
taxi[1..5] = Taxi ELEMENT

Auto = Vehicle ELEMENT TYPE
auto[1..15] = Auto ELEMENT

Restaurant Lobby and Reservations

```
lobby = ELEMENT
EVENTS
  Enter(f:Friend)
END lobby
```

```
reservations = ELEMENT
EVENTS
  Reserve(p:Party, b:Bribe)
  Seat(p:Party, t:RestaurantTable)
CONSTRAINTS
```

- 1) To be seated, there must be a reservation. Moreover, each reservation is good for only one seating.
 $\text{Reserve}(p,b) \longrightarrow \text{Seat}(p,t)$
 - 2) Parties can only be seated at tables of the right size.
 $\text{occur}(\text{seat}) \supset \text{seat.p.size} = \text{seat.t.size}$
 - 3) A bigger bribe will get you seated faster.
 $\text{reserve1 cbefore Seat} \wedge \text{reserve2 cbefore Seat} \wedge \text{reserve1.b} > \text{reserve2.b} \supset$
 $\square [(\exists \text{seat2:Seat}) \text{reserve2} \leadsto \text{seat2} \supset (\exists \text{seat1:Seat}) \text{reserve1} \leadsto \text{seat1}]$
 - 4) All other things being equal, seating is first-come-first-served.
 $\text{reserve1}(p1,b1) \implies \text{reserve2}(p2,b2) \wedge b1=b2 \wedge$
 $\text{present}(p1) \wedge \text{present}(p2) \supset$
 $\square [(\exists \text{seat2:Seat}) \text{reserve2} \leadsto \text{seat2} \supset (\exists \text{seat1:Seat}) \text{reserve1} \leadsto \text{seat1}]$
 - 5) A \$50 bribe will definitely get you seated.
 $\text{reserve.b} \geq \$50 \supset \Diamond (\exists \text{seat:Seat}) \text{reserve} \leadsto \text{seat}$
- ```
END reservations
```

where

$\text{present}(p) \equiv (\forall f:\text{Friend}, f \neq p) (\exists \text{enter}(f):\text{Lobby.Enter})$   
 $\text{enter}(f) \text{ cbefore Reservations.Seat}$

## The Friends

Friend=GROUP TYPE (m:Movement,c:Communication,f:FoodOrdering,a:Auto)

### CONSTRAINTS

- 1) Each friend must sit and order before they can eat.  
 $m.Sit \longrightarrow c.OrderFood \longrightarrow c.Eat$
- 2) If a friend is eating, they must still be sitting at a table.  
 $justoccurred(c.eat) \wedge m.sit \sim c.orderfood \sim c.eat \supset$   
 $m.sit \text{ cbefore } m.LeaveTable$
- 3) To ride somewhere, a friend may use their own car.  
 $m.Ride(loc1,loc2) \doteq a.Occupy \longrightarrow a.Drive(loc1,loc2) \longrightarrow a.Vacate$
- 4) To order food, a friend must first read the menu, choose a meal, and then tell the waiter.  
 $c.OrderFood \doteq f.f.s.Read(menu) \longrightarrow f.t.Choose(food) \longrightarrow f.s.Utter(food)$

END Friend

Movement = ELEMENT TYPE

### EVENTS

Ride(loc1,loc2:Location)  
 Walk(loc1,loc2:Location)  
 Sit(tableloc:Location)  
 LeaveTable(tableloc,loc:Location)

### CONSTRAINTS

- 1) A friend must walk to a table before sitting there, and must be sitting there before leaving.  
 $Walk(loc1,tableloc) \longrightarrow Sit(tableloc) \longrightarrow LeaveTable(tableloc,loc2)$
- 2) To depart from a location x, a friend must first be there.  
 $(\forall move(x,y): \{Ride, Walk, LeaveTable\}) \text{ Precondition}(move(x,y), at(x))$

$at(x) \equiv$

$(INIT \ at(x) \wedge \neg (\exists move: \{Ride, Walk, LeaveTable\}) \text{ occurred}(move)) \vee$   
 $((\exists move(z,x): \{Ride, Walk, LeaveTable\}) \text{ occurred}(move(z,x)) \wedge$   
 $\neg (\exists move': \{Ride, Walk, LeaveTable\}) \text{ move}(z,x) \implies move')$

END Movement

Communication = ELEMENT TYPE

EVENTS

Reserve(p:Party, b:Bribe)

OrderFood(food:Foodstuff)

Eat

END Communication

FoodOrdering = GROUP TYPE (t:Thought, s:Speech, r:Sight, f:Order)  
PORTS(f.OrderFood', f.OrderFood")

Thought = ELEMENT TYPE

EVENTS

Choose(food:Foodstuff)

END Thought

Speech = ELEMENT TYPE

EVENTS

Utter(food:Foodstuff)

END Speech

Sight = ELEMENT TYPE

EVENTS

Read(menu:ReadingMaterial)

END Sight

Order = ELEMENT TYPE

EVENTS

OrderFood'(food:Foodstuff)

OrderFood"(food:Foodstuff)

END Order

comm[1..15] = Communication ELEMENT  
 move[1..15] = Movement ELEMENT  
 thought[1..15] = Thought ELEMENT  
 speech[1..15] = Speech ELEMENT  
 sight[1..15] = Sight ELEMENT  
 order[1..15] = Order ELEMENT  
 foodorder[i=1..15] = FoodOrder GROUP (thought[i],speech[i],sight[i],order[i])  
 auto[1..15] = Auto ELEMENT  
 friend[i=1..15] = Friend GROUP (comm[i],move[i],foodorder[i],auto[i])

### *The Parties*

Party(size:INTEGER) =  
   GROUP TYPE ({f}:SET OF Friend, reservations,lobby)  
 CONSTRAINTS  
 1) (size must be the size of the set of friends)  
    size = setsize({f})  
 2) A reservation by a friend is identified with a reservation at the desk.  
    f.c.Reserve(p,b)  $\approx$  reservations.Reserve(p,b)  
 3) All members of a party must be seated simultaneously.  
    ( $\forall f' \in \{f\}$ ) reservations.Seat(SELF,t)  $\sim$  f'.m.Sit(t)  
 4) In order to be seated, all members of the party must be present.  
    ( $\forall f' \in \{f\}$ ) lobby.Enter(f')  $\longrightarrow$  reservations.Seat(SELF,Table)  
 5) The first friend to enter the lobby must make a reservation.  
    ( $\forall f1,f2 \in \{f\}$ ) occurred(lobby.enter1(f1))  $\wedge$   
        $\neg(\exists \text{ lobby.enter2}(f2)) [\text{lobby.enter2}(f2) \implies \text{lobby.enter1}(f1)] \supset$   
        $\Diamond (\exists \text{ reserve1}:f1.c.\text{Reserve}(\text{SELF},b)) \text{ occurred}(\text{reserve1})$   
 END Party  
  
 party[1] = GROUP ({friend[1..3]},reservations,lobby)  
 party[2] = GROUP ({friend[4..8]},reservations,lobby)  
 party[3] = GROUP ({friend[9..15]},reservations,lobby)

### *Felix*

```
observations = ELEMENT
EVENTS
 EmptyTable(t)
END observations
```

```
felixworld = GROUP (reservations, table[1..10], lobby, observations)
 PORTS(table[i].Vacate)
```

#### CONSTRAINTS

- 1) Felix must observe that a table is empty before he can seat a party there.  
observations.EmptyTable(table[i])  $\longrightarrow$  reservations.Seat(Party,table[i])
- 2) Seating a party at a table is the same as occupying the table.  
reservations.Seat(p,table[i])  $\approx$  table[i].Occupy(p)
- 3) Felix must make correct observations about empty tables.  
( $\forall$  empty(t):observations.EmptyTable) Precondition(empty(t),TableEmpty(t))  
TableEmpty(table[i])  $\equiv$   
 $\neg (\exists$  occupy:table[i].Occupy) occurred(occupy)  $\vee$   
( $\exists$  vacate:table[i].Vacate) vacate cbefore table[i].Occupy

```
END felixworld
```

### *The Entire Restaurant Scenario*

```
restaurantscenario = GROUP(felixworld, party[1..3], taxi[1..5])
CONSTRAINTS
1) ($\forall$ taxi[i],party[j])
 party[j].f.m.Ride(loc1,loc2) $\doteq$
 taxi[i].Occupy $\longrightarrow$ taxi[i].Drive(loc1,loc2) $\longrightarrow$ taxi[i].Vacate
END restaurantscenario
```

## Chapter 6

# EMBEDDED PLANNING SYSTEMS FOR MULTIAGENT DOMAINS

*This work will appear in the proceedings of the AAAI Conference in Seattle, Washington in August, 1987. It was written by Amy Lansky and Michael Georgeff.*

### A Introduction

The ability to act appropriately in dynamic environments is critical for the survival of all living creatures. For lower life forms, it seems that sufficient capability is provided by stimulus-response and feedback mechanisms. Higher life forms, however, must be able to anticipate future events and situations, and form plans of action to achieve their goals. The design of reasoning and planning systems that are *embedded* in the world and must operate effectively under real-time constraints can thus be seen as fundamental to the development of intelligent autonomous machines.

In this paper, we describe a system for reasoning about and performing complex tasks in dynamic environments, and show how it can be applied to the control of an autonomous mobile robot. The system, called a *Procedural Reasoning System* (PRS), is endowed with the attitudes of belief, desire, and intention. At any given instant, the actions being considered by PRS depend not only on its current desires or goals, but also on its beliefs and previously formed intentions. PRS also has the ability to reason about its own internal state – that is, to reflect upon its own beliefs, desires, and intentions, modifying these as it chooses. This architecture allows PRS to reason about means and ends in much the same way as do traditional planners, but provides the reactivity that is essential for survival in highly dynamic and uncertain worlds.

For our the task domain, we envisaged a robot in a space station, fulfilling the role of an astronaut's assistant. When asked to get a wrench, for example, the robot determines

where the wrench is kept, plans a route to that location, and goes there. If the wrench is not where expected, the robot may reason further about how to obtain information as to its whereabouts. It then either returns to the astronaut with the desired tool or explains why it could not be retrieved. In another scenario, the robot may be midway through the task of retrieving the wrench when it notices a malfunction light for one of the jets in the reactant control system of the space station. It reasons that handling this malfunction is a higher-priority task than retrieving the wrench and therefore sets about diagnosing the fault and correcting it. Having done this, it resumes its original task, finally telling the astronaut.

To accomplish these tasks, the robot must not only be able to create and execute plans, but must be willing to interrupt or abandon a plan when circumstances demand it. Moreover, because the robot's world is continuously changing and other agents and processes can issue demands at arbitrary times, performance of these tasks requires an architecture that is both highly reactive and goal-directed.

We have used PRS with the new SRI robot, Flakey, to exhibit much of the behavior described in the foregoing scenarios, including both the navigational and malfunction-handling tasks [36]. In this paper, we concentrate on the navigational task; the knowledge base used for jet malfunction handling is described elsewhere [34,35].

## B Previous Approaches

Most existing architectures for embedded planning systems consist of a plan constructor and a plan executor. As a rule, the plan constructor formulates an entire course of action before commencing execution of the plan [25,114,116]. The plan itself is typically composed of primitive actions – that is, actions that are directly performable by the system. The rationale for this approach, of course, is to ensure that the planned sequence of actions will actually achieve the prescribed goal. As the plan is executed, the system performs these primitive actions by calling various low-level routines. Execution is usually monitored to ensure that these routines will culminate in the desired effects; if they do not, the system can return control to the plan constructor so that it may modify the existing plan appropriately.

One problem with these schemes is that, in many domains, much of the information about how best to achieve a given goal is acquired during plan execution. For example, in planning to get from home to the airport, the particular sequence of actions to be performed depends on information acquired on the way – such as which turnoff to take, which lane to get into, when to slow down or speed up, and so on. To overcome this problem, at least in part, there has been some work on developing planning systems that interleave plan formation and execution [18,20]. Such systems are better suited to uncertain worlds than the kind of system described above, as decisions can be deferred until they *have* to be made. The reason for deferring decisions is that an agent can acquire *more* information as time passes; thus, the quality of its decisions can be expected only to improve. Of course, because of the need to coordinate some activities in advance and because of practical restrictions on the amount of decision-making that can be

accommodated during task execution, there are limitations on the degree to which such decisions may be deferred.

Real-time constraints pose yet further problems for traditionally structured systems. First, the planning techniques typically used by these systems are very time-consuming, requiring exponential search through potentially enormous problem spaces. While this may be acceptable in some situations, it is not suited to domains where replanning is frequently necessary and where system viability depends on readiness to act.

In addition, most existing systems are overcommitted to the planning phase of their operations; no matter what the situation or how urgent the need for action, these systems *always* spend as much time as necessary to plan and reason about achieving a given goal before performing any external actions whatsoever. They lack the ability to decide when to stop planning or to reason about possible compromises between further planning and longer available execution time.

Traditional planning systems also rely excessively on constructing plans solely from knowledge about the primitive actions performable by the robot. However, many plans are not constructed from first principles, but have been acquired in a variety of other ways – for example, by being told, by learning, or through training. Furthermore, these plans may be very complex, involving a variety of control constructs (such as iteration and recursion) that are normally not part of the repertoire of conventional planning systems. Thus, although it is obviously desirable that an embedded system be capable of forming plans from first principles, it is also important that the system possess a wealth of precompiled *procedural knowledge* about how to function in the world [34].

The real-time constraints imposed by dynamic environments also require that a situated system be able to react quickly to environmental changes. This means that the system should be able to *notice* critical changes in the environment within an appropriately small interval of time. However, most embedded planning systems provide no mechanisms for reacting in a timely manner to new situations or goals during plan execution, let alone during plan formation.

Another disadvantage of most systems is that they commit themselves strongly to the plans they have adopted. While such systems may be reactive in the limited sense of being able to replan so as to accomplish fixed goals, they are unable to change their focus completely and pursue new goals when the situation warrants. Indeed, the very survival of an autonomous system may depend on its ability to modify its goals and intentions according to the situation.

A number of systems developed for the control of robots do have a high degree of reactivity [2]. Even SHAKEY [86] utilized reactive procedures (ILAs) to realize the primitive actions of the high-level planner (STRIPS). This idea is pursued further in some recent work by Nilsson [87]. Another approach is advocated by Brooks [9], who proposes decomposition of the problem into *task-achieving* units whereby distinct behaviors of the robot are realized separately, each making use of the robot's sensors, effectors, and reasoning capabilities as needed. Kaelbling [49] proposes an interesting hybrid architecture based on similar ideas.

These kinds of architectures could lead to more viable and robust systems than the traditional robot-control systems. Yet most of this work has not addressed the issues of

general problem-solving and commonsense reasoning; the research is instead almost exclusively devoted to problems of navigation and the execution of low-level actions. These techniques have yet to be extended or integrated with systems that can change goal priorities completely, modify, defer, or abandon its plans, and reason about what is best to do in light of the immediate situation.

In sum, existing planning systems incorporate many useful techniques for constructing plans of action in a great variety of domains. However, most approaches to embedding these planners in dynamic environments are not robust enough nor sufficiently reactive to be useful in many real-world applications. On the other hand, the more reactive systems developed in robotics are well suited to handling the low-level sensor and effector activities of a robot. Nevertheless, it is not yet clear how these techniques could be used for performing some of the higher-level reasoning desired of complex problem-solving systems. To reconcile these two extremes, it is necessary to develop reactive reasoning and planning systems that can utilize both kinds of capabilities whenever they are needed.

## C A Reactive Planning System

The system we used for controlling and carrying out the high-level reasoning of the robot is called a *Procedural Reasoning System* (PRS) [34,35]. The system consists of a *data base* containing current *beliefs* or facts about the world, a set of current *goals* or *desires* to be realized, a set of *procedures* (which, for historical reasons, are called *knowledge areas* or KAs) describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations, and an *interpreter* (or *inference mechanism*) for manipulating these components. At any moment, the system will also have a *process stack* (containing all currently active KAs) which can be viewed as the system's current *intentions* for achieving its goals or reacting to some observed situation. The basic structure of PRS is shown in Figure 6.1. A brief description of each component and its usage is given below.

### C.1 The System Data Base

The contents of the PRS data base may be viewed as representing the current beliefs of the system. Some of these beliefs may be provided initially by the system user. Typically, these will include facts about static properties of the application domain — for example, the structure of some subsystem, or the physical laws that some mechanical components must obey. Other beliefs are derived by PRS itself as it executes its KAs. These will typically be current observations about the world or conclusions derived by the system from these observations.

The data base itself consists of a set of *state descriptions* describing what is believed to be true at the current instant of time. We use first-order predicate calculus for the state description language. Data base queries are handled using unification over the set of data base facts. State descriptions that describe *internal* system states are called *metalevel*

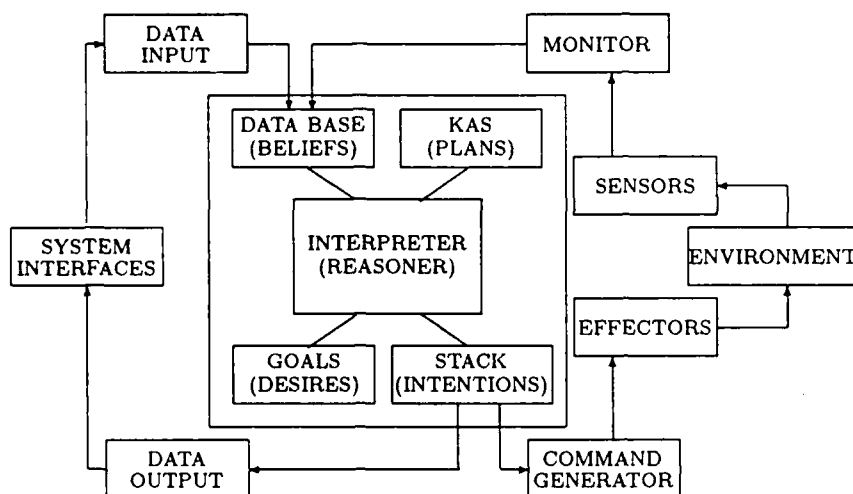


Figure 6.1: System Structure

expressions. The basic metalevel predicates and functions are predefined by the system. For example, the metalevel expression `(goal g)` is true if `g` is a current goal of the system.

## C.2 Goals

Goals appear both on the system goal stack and in the representation of KAs. Unlike most AI planning systems, PRS goals represent desired *behaviors* of the system, rather than static world states that are to be [eventually] achieved. Hence goals are expressed as conditions on some interval of time (i.e., on some sequence of world states).

Goal behaviors may be described in two ways. One is to apply a *temporal predicate* to an  $n$ -tuple of terms. Each temporal predicate denotes an *action type* or a *set* of state sequences. That is, an expression like `(walk a b)` can be considered to denote the set of state sequences which embody walking actions from point `a` to `b`.

A behavior description can also be formed by applying a temporal operator to a state description. Three temporal operators are currently used. The expression `(!p)`, where `p` is some state description (possibly involving logical connectives), is true of a sequence of states if `p` is true of the last state in the sequence; that is, it denotes those behaviors that *achieve* `p`. Thus we might use the behavior description `(!(walked a b))` rather than `(walk a b)`. Similarly, `(?p)` is true if `p` is true of the first state in the sequence – that is, it can be considered to denote those behaviors that result from a successful *test* for `p`. Finally, `(#p)` is true if `p` is preserved (maintained invariant) throughout the sequence. Behavior descriptions can be combined using the logical operators  $\wedge$  and  $\vee$ . These denote, respectively, the intersection and union of the composite behaviors.

As with state descriptions, behavior descriptions are not restricted to describing the external environment, but can also be used to describe the internal behavior of the system. Such behavior specifications are called metalevel behavior specifications. One important metalevel behavior is described by an expression of the form  $(\Rightarrow p)$ . This specifies a behavior that places the state description  $p$  in the system data base. Another way of describing this behavior might be  $(!(\text{belief } p))$ .

### C.3 Knowledge Areas

Knowledge about how to accomplish given goals or react to certain situations is represented in PRS by declarative procedure specifications called *Knowledge Areas* (KAs). Each KA consists of a *body*, which describes the steps of the procedure, and an *invocation condition* that specifies under what situations the KA is useful.

The body of a KA is represented as a graphic network and can be viewed as a plan or plan schema. However, it differs in a very important way from the plans produced by most AI planners: it does not consist of possible sequences of primitive actions, but rather of possible sequences of *subgoals* to be achieved. Thus, the bodies of KAs are much more like the high-level "operators" used in traditional planning systems [116]. They differ in that (1) the subgoals appearing in the body can be described by complex temporal expressions and (2) the allowed control constructs are richer and include conditionals, loops, and recursion.

The invocation part of a KA contains an arbitrarily complex logical expression describing under what conditions the KA is useful. Usually this consists of some conditions on current system goals (in which case, the KA is invoked in a goal-directed fashion) or current system beliefs (resulting in data-directed or *reactive* invocation), and may involve both. Together the invocation condition and body of a KA express a declarative fact about the effects of performing certain sequences of actions under certain conditions.

The set of KAs in a PRS application system not only consists of procedural knowledge about a specific domain, but also includes *metalevel* KAs — that is, information about the manipulation of the beliefs, desires, and intentions of PRS itself. For example, typical metalevel KAs encode various methods for choosing among multiple relevant KAs, determining how to achieve a conjunction of goals, and computing the amount of additional reasoning that can be undertaken, given the real-time constraints of the problem domain. Metalevel KAs may of course utilize knowledge specifically related to the problem domain. In addition to user-supplied KAs, each PRS application contains a set of system-defined default KAs. These are typically domain-independent metalevel KAs.

### C.4 The System Interpreter

The PRS interpreter runs the entire system. From a conceptual standpoint, it operates in a relatively simple way. At any particular time, certain goals are active in the system and certain beliefs are held in the system data base. Given these extant goals and beliefs, a

subset of KAs in the system will be relevant (i.e., applicable). One of these relevant KAs will then be chosen for execution by placing it on the process stack.

In the course of executing the chosen KA, new subgoals will be posted and new beliefs derived. When new goals are pushed onto the goal stack, the interpreter checks to see if any new KAs are relevant, chooses one, places it on the process stack, and begins executing it. Likewise, whenever a new belief is added to the data base, the interpreter will perform appropriate consistency maintenance procedures and possibly activate other relevant KAs. During this process, various metalevel KAs may also be called upon to make choices among alternative paths of execution, choose among multiple applicable KAs, decompose composite goals into achievable components, and make other decisions.

This results in an interleaving of plan selection, formation, and execution. In essence, the system forms a partial overall plan, determines a means of accomplishing the first subgoal of the plan, acts on this, further expands the near-term plan of action, executes further, and so on. At any time, the plans the system is intending to execute (i.e., the selected KAs) are both *partial* and *hierarchical* — that is, while certain general goals have been decided upon, the specific means for achieving these ends have been left open for future deliberation.

Unless some new fact or request activates some new KA, PRS will try to fulfill any intentions it has previously decided upon. But if some important new fact or request does become known, PRS will reassess its goals and intentions, and then perhaps choose to work on something else. Thus, not all options that are considered by PRS arise as a result of means-end reasoning. Changes in the environment may lead to changes in the system's beliefs, which in turn may result in the consideration of new plans that are not means to any already intended end. PRS is therefore able to *change its focus completely* and pursue new goals when the situation warrants it. PRS can even alter its intentions regarding its own reasoning processes — for example, it may decide that, given the current situation, it has no time for further reasoning and so must act immediately.

### C.5 Multiple Asynchronous PRSs

In some applications, it is necessary to monitor and process many sources of information at the same time. Because of this, PRS was designed to allow several instantiations of the basic system to run in parallel. Each PRS instantiation has its own data base, goals, and KAs, and operates asynchronously relative to other PRS instantiations, communicating with them by sending messages. The messages are written into the data base of the receiving PRS, which must then decide what to do, if anything, with the new information. As a rule, this decision is made by a fact-invoked KA (in the receiving PRS), which responds upon receipt of the external message. In accordance with such factors as the reliability of the sender, the type of message, and the beliefs, goals, and current intentions of the receiver, it is determined what to do about the message — for example, to acquire a new belief, establish a new goal, or modify intentions.

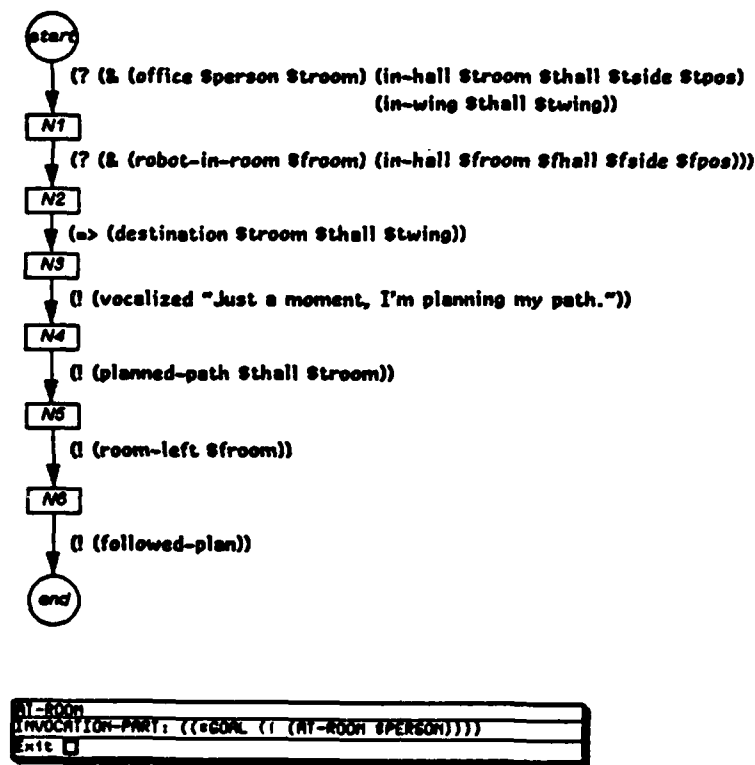


Figure 6.2: The Top-Level Strategy

## D The Domain Knowledge

The scenario described in the introduction includes problems of route planning, navigation to maintain the route, and such tasks as malfunction handling and requests for information. We shall concentrate herein on the tasks of route planning and navigation. However, it is important to realize that the knowledge representation provided by PRS is used for reasoning about all tasks performed by the system.

The way the robot (under the control of PRS) solves the tasks of the space station scenario is roughly as follows. To reach a particular destination, it knows that it must first plan a route and then navigate to the desired location (see the KA depicted in Figure 6.2). In planning the route, the robot uses knowledge of the station's topology to work out a path to the target location, as is typically done in navigational tasks for autonomous robots. The topological knowledge is not detailed, stating simply which rooms are in which corridors and how the latter are connected. The route plan formed by the robot is also high-level, typically having the following form: "Travel to the end of the corridor, turn right, then go to the third room on the left." The robot's knowledge of the problem domain's topology is stored in its data base, while its knowledge of how to plan a route is represented in various route-planning KAs. Throughout this predictive-planning stage, the robot remains continuously reactive. Thus, for example, should the robot notice indication

of a jet failure on the space station, it may well decide to interrupt its route planning and attend instead to the task of remedying the jet problem.

Once a plan is formed by the route-planning KAs, that plan must be used to guide the activities of the robot. To achieve this, we defined a group of KAs that react to the presence of a plan (in the data base) by translating it into the appropriate sequence of subgoals. Each leg of the original route plan generates subgoals – such as turning a corner, travelling along the hallway, and updating the data base to indicate progress. The second group of navigational KAs reacts to these goals by actually doing the work of reading the sonars, interpreting the readings, counting doorways, aligning the robot in the hallway, and watching for obstacles up ahead.

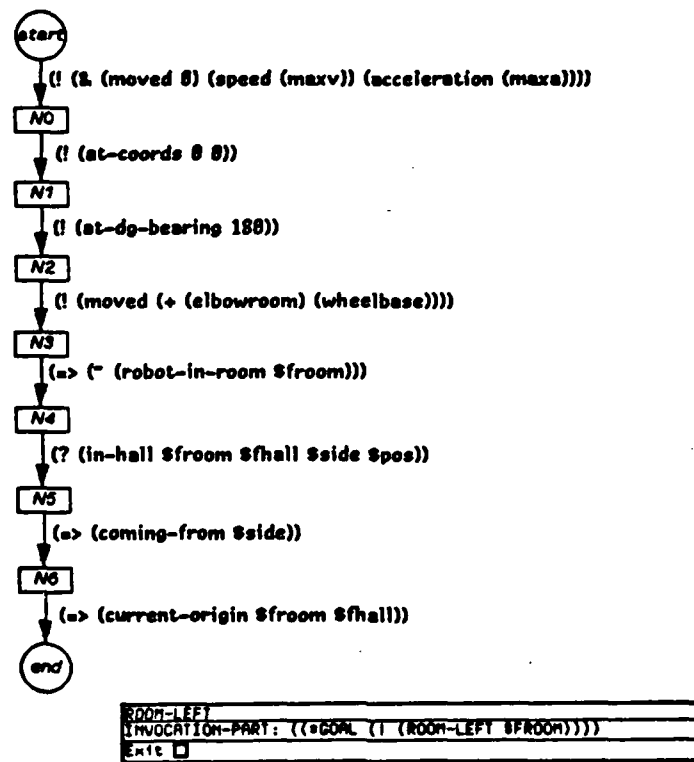


Figure 6.3: Route Navigation KA

For example, let us consider the KAs in Figures 6.3 and 6.4. After having used the KA in Figure 6.2 to plan a path, the robot acquires the goal (! (room-left \$froom)), where the variable \$froom is bound to some particular constant representing the room that the robot is trying to leave. The KA in Figure 6.3 will respond, causing the robot to perform the steps for leaving the given room. The last step in this KA will insert a fact into the system data base of the form (current-origin \$froom \$fhall), where the variables are again bound to specific constants. Next, the KA in Figure 6.2 issues the command (! (follow-plan)). This activates the KA in Figure 6.4, which assures that each leg of the plan is followed until the goal destination is reached. Beliefs of the form

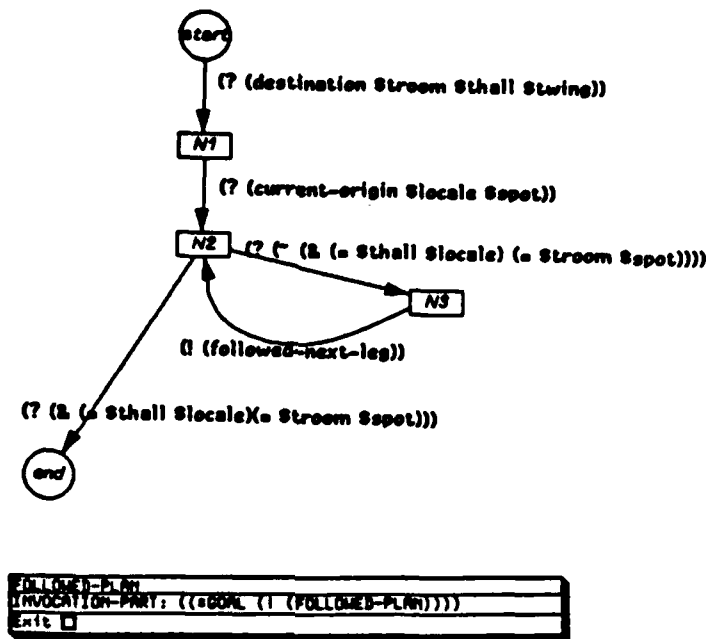


Figure 6.4: Plan Interpretation KA

(current-origin \$locale \$spot) are repeatedly updated to readjust the robot's bearings and knowledge about its whereabouts.

A third group of KAs reacts to contingencies encountered by the robot as it interprets and follows its path. These will include KAs that respond to the presence of an obstacle ahead or the fact that an emergency light has been seen. Such reactive KAs are invoked solely on the basis of certain facts' becoming known to the robot. Implicit in their invocation, however, is an underlying goal to "avoid obstacles" or "remain safe."

Yet other KAs perform the various other tasks required of the robot [35]. Metalevel KAs choose among different means of realizing any given goal and determine the respective priority of tasks when mutually inconsistent goals arise (such as diagnosing a jet failure and fetching a wrench). Each KA manifests a self-contained behavior, possibly including both sensory and effector components. Many of these KAs can be simultaneously active, performing their function whenever they may be applicable. Thus, while trying to follow a path down a hallway, an obstacle avoidance procedure may simultaneously cause the robot to veer from its original path. We elsewhere provide a more detailed description of the KAs used by the robot [36].

## E Discussion

The system as described here was implemented using the new SRI robot, Flakey, to accomplish much of the two scenarios described in the introduction. In particular, the robot managed to plan a path to the target room, maneuver its way out of the room in which it was stationed, and navigate to its destination via a variety of hallways,

intersections, and corners. It maintained alignment in the hallways, avoided obstacles, and stopped whenever its path was completely blocked. If it noticed a jet malfunction on the space station (simulated by human interaction via the keyboard), it would interrupt whatever it was doing (route planning, navigating the hallways, etc.) and attend to diagnosing the problem. The diagnosis performed by the robot was quite complex and followed actual procedures used for NASA's space shuttle [35].

The features of PRS that, we believe, contributed most to this success were (1) its partial planning strategy, (2) its reactivity, (3) its use of procedural knowledge, and (4) its metalevel (reflective) capabilities. The partial hierarchical planning strategy and the reflective reasoning capabilities of PRS proved to be well suited to the robot application, yet still allowed the system to plan ahead when necessary. By finding and executing relevant procedures only when sufficient information was available, the system stood a better chance of achieving its goals under the stringent real-time constraints of the domain. For example, the method for determining the robot's course was dynamically influenced by the situation, such as whether the robot was between two hallway walls, adjacent to an open door, at a T-intersection, or passing an unknown obstacle.

Because PRS expands plans dynamically and incrementally, there were also frequent opportunities for it to react to new situations and changing goals. For example, when the system noticed a jet-fail alarm while it was attempting to fetch a wrench, it had the ability to reason about the priorities of these tasks and, if it so decided, to suspend the wrench-fetching task while it attended to the jet failure. Indeed, the system even continued to monitor the world while it was *planning* its route and could interrupt the planning whenever the situation demanded.

The wealth of procedural knowledge possessed by the system was also critical in allowing the robot to operate effectively in real-time and to perform a variety of very complex tasks. In particular, the powerful control constructs allowed in KAs (such as conditionals, loops, and recursion) proved highly advantageous. PRS also makes it possible to have a large number of diverse KAs available for achieving a goal. Each may vary in its ability to accomplish a goal, as well as in its applicability in particular situations. Thus, if there is insufficient information about a given situation to allow one KA to be used, another (perhaps one less reliable) might be available instead. Parallelism and reactivity also helped in providing robustness. For example, if one PRS instantiation were busy planning a route, other instantiations could remain active, monitoring environmental changes, keeping the robot in a stable configuration, and avoiding dangers. This has much in common with, and yields the same advantages as, the vertical robot architecture proposed by Brooks [9].

The metalevel reasoning capabilities of PRS were particularly important in managing the application of the various KAs in different situations. Such capabilities can be critical in deciding how best to meet the real-time constraints of a domain. However, the current system was really too simple to serve as an adequate test of the system's metalevel reasoning abilities; indeed, the system performed quite well with only a few [well-chosen] metalevel KAs.

Despite these encouraging results, the research is only in its initial stages and there are a

number of limitations that still need to be addressed. First, there are many assumptions behind the procedures (KAs) used. For example, we have assumed that hallways are straight and corners rectangular and that all doors are open and unobstructed. A greater variety of KAs and increased parallelism would also have been preferable, allowing the robot to perform its tasks under more demanding conditions. For example, we could have included many additional low-level procedures for, say, avoiding dangers and exploring the surroundings. Finally, PRS does not reason about other subsystems (i.e., other PRS instantiations) in any but the simplest ways. However, the message-passing mechanisms we have employed should allow us to integrate more complex reasoning about interprocess communication.

## Acknowledgments

Marcel Schoppers carried out the experiment described here. Pierre Bessiere, Joshua Singer, and Mabry Tyson helped in the development of PRS. Stan Reifel and Sandy Wells designed Flakey and its interfaces, and assisted with the implementation described herein. We have also benefited from our participation and interactions with members of CSLI's Rational Agency Group (RATAG), particularly Michael Bratman, Phil Cohen, Kurt Konolige, David Israel, and Martha Pollack. Leslie Pack Kaelbling, Stan Rosenschein, and Dave Wilkins also provided helpful advice and interesting comments.

# Bibliography

- [1] Abadi, M. and Manna, Z. (1985). Nonclausal temporal deduction. Report No. STAN-CS-85-1056, Computer Science Department, Stanford University, Stanford, California.
- [2] J. S. Albus. *Brains, Behavior, and Robotics*. McGraw-Hill, Peterborough, New Hampshire, 1981.
- [3] Allen, J. F., "Towards a General Theory of Action and Time," *Artificial Intelligence*, 23, pp. 123-154 (1984).
- [4] Allen, J. F. *Recognizing Intentions from Natural Language Utterances*, pages 107-166. MIT Press, Cambridge, Massachusetts, 1983.
- [5] Allen, J.F. and J.A.Koomen, "Planning Using a Temporal World Model," *IJCAI-83, Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp. 741-747 (August 1983).
- [6] Appelt, D. (1985). *Planning English sentences*. Cambridge University Press, Cambridge, U. K.
- [7] Barringer, H. and R. Kuiper, "Hierarchical Development of Concurrent Systems in a Temporal Logic Framework," *Proceedings of the NSF/SERC Seminar on Concurrency*, Carnegie Mellon University, Pittsburgh, Pennsylvania (July 1984).
- [8] Bratman, M., *Intention, Plans, and Practical Reason*, Harvard University Press, Cambridge, Massachusetts, forthcoming.
- [9] R. A. Brooks. *A Robust Layered Control System for a Mobile Robot*. Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1985.
- [10] Chapman, D. "Cognitive Cliches," AI Working Paper 286, MIT Laboratory for Artificial Intelligence, Cambridge, Massachusetts (April 1986).
- [11] Chapman, D. "Planning for Conjunctive Goals," Masters Thesis, Technical Report MIT-AI-TR-802, MIT Laboratory for Artificial Intelligence, Cambridge, Massachusetts (1985).

- [12] Cheeseman, P. "A Representation of Time for Automatic Planning," *Proceedings of the IEEE International Conference on Robotics*, Atlanta, Georgia (March 1984).
- [13] Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science* 3, pp. 177-212.
- [14] Cohen, P. and Levesque, H. Persistence, intention, and commitment. In *Proceedings of the Workshop on Planning and Reasoning about Action*, Timberline, Oregon, Morgan Kaufmann, 1986.
- [15] Cohen, P. R. and Levesque, H. J. Speech acts and the recognition of shared plans. In *Proceedings of the Third Biennial Conference*, pages 263-271, Canadian Society for Computational Studies of Intelligence, 1980.
- [16] Cohen, P. R. and H. J. Levesque, "Speech Acts and the Recognition of Shared Plans," *Proceedings of the Twenty Third Conference of the Association for Computational Linguistics*, Stanford, California (1985).
- [17] Davidson, D. *Essays on Actions and Events*, Clarendon Press, Oxford, England (1980).
- [18] P.R. Davis and R.T. Chien. Using and reusing partial plans. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 494, Cambridge, Massachussets, 1977.
- [19] Dean, T., "Planning and Temporal Reasoning under Uncertainty," *Proceedings of the IEEE Workshop on Knowledge Based Systems*, Denver, Colorado (1984).
- [20] E. H. Durfee and V. R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58-64, Philadelphia, Pennsylvania, 1986.
- [21] Etherington, D. W. *Reasoning with Incomplete Information: Investigations of Non-Monotonic Reasoning*. PhD thesis, University of British Columbia, Vancouver, British Columbia, 1986.
- [22] Fahlman, S. E. *NETL: A System For Representaing and Using Real-World Knowledge*. MIT Press, Cambridge, Massachusetts, 1981.
- [23] Fagin, R. and Halpern, J. Y. (1985). Belief, awareness, and limited reasoning. In *Proceedings of the Ninth International Joint Conference on AI*, Los Angeles, California, pp. 491-501.
- [24] Fariñas-del-Cerro, L. (1983). Temporal reasoning and termination of programs. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp. 926-929.

- [25] Fikes, R. E., and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, 2, pp. 189 - 208 (1971).
- [26] Fox, M.S. and Smith, S.F. "ISIS - A Knowledge-Based System for Factory Scheduling," *Expert Systems, the International Journal of Knowledge Engineering*. Volume 1, Number 1, pp. 25-49 (July 1984).
- [27] Geissler, C. and Konolige, K. (1986). Implementation of a resolution system for modal logic. Forthcoming Artificial Intelligence Center Tech Note, SRI International, Menlo Park, California.
- [28] Georgeff, M. P. "Communication and Interaction in Multitagent Planning," *Proceedings of the Third National Conference on Artificial Intelligence*, Washington, D.C. (1983).
- [29] Georgeff, M. P., "A Theory of Action for Multiagent Planning," *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, Texas (1984).
- [30] Georgeff, M. P., "Reasoning about Procedural Knowledge," *Proceedings of the AIAA/ACM/NASA/IEEE Computers in Aerospace Conference*, Long Beach, California (1985).
- [31] Georgeff, M. P., "Actions, Processes, and Causality," Artificial Intelligence Center Technical Note, SRI International, Menlo Park, California (1986).
- [32] Georgeff, M. P., "A Representation of Events in Multiagent Domains," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania (1987).
- [33] Georgeff, M. P., "Many Agents are Better than One," Forthcoming SRI Technical Note, Artificial Intelligence Center, SRI International, Menlo Park, California. (1987).
- [34] Georgeff, M. P., and Lansky, A. L., "Procedural Knowledge," *Proc. IEEE*, Special Issue on Knowledge Representation (1986).
- [35] M. P. Georgeff and A. L. Lansky. *A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle*. Technical Note 375, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986.
- [36] M. P. Georgeff, A. L. Lansky, and M. Schoppers. *Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot*. Technical Note 380, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.
- [37] Grosz, B. J. (1981). Focusing and description in natural language dialogues. In *Elements of Discourse Understanding*, Cambridge University Press, Joshi, A. K., Webber, B., and Sag, I., Eds.

- [38] Halpern, J. Y. and Moses, Y. (1984). Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd ACM Conference on Principles of Distributed Computing*, pp. 50-61.
- [39] Halpern, J. Y. and Moses, Y. (1985). A guide to the modal logics of knowledge and belief: preliminary draft. In *Proceedings of the Ninth International Joint Conference on AI*, Los Angeles, California, pp. 479-490.
- [40] Hanks, S., and McDermott, D., "Default Reasoning, Nonmonotonic Logics, and the Frame Problem," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania (1986).
- [41] Harel, D. *First Order Dynamic Logic*, Lecture Notes in Computer Science, 68, Springer-Verlag (1979).
- [42] Hayes, P. J., "The Frame Problem and Related Problems in Artificial Intelligence," in *Artificial and Human Thinking*, A. Elithorn and D. Jones (eds.), Jossey-Bass (1973).
- [43] Hewitt, C. and H. Baker Jr. "Laws for Communicating Parallel Processes," *IFIP 77*, B.Gilchrist,ed., pp. 987-992, North-Holland, Amsterdam, Holland (1977).
- [44] Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press, Ithaca, New York.
- [45] Hintikka, J. (1969). Semantics for propositional attitudes. In L. Linsky (ed.), *Reference and Modality*, Oxford University Press, London (1971), pp. 145-167.
- [46] Hintikka, J. Impossible possible worlds vindicated. *Journal of Philosophical Logic*, 4:475-484, 1975.
- [47] Hoare, C. A. R., *Communicating Sequential Processes*, Series in Computer Science, C. A. R. Hoare (ed.), Prentice Hall, Englewood Cliffs, New Jersey (1985).
- [48] Imielinski, T. Results on translating defaults to circumscription. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 114-120, Los Angeles, 1985.
- [49] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.
- [50] Konolige, K. (1980). A first-order formalization of knowledge and action for a multiagent planning system. Artificial Intelligence Center Tech Note 232, SRI International, Menlo Park, California.
- [51] Konolige, K. (1984). A deduction model of belief and its Logics. Doctoral dissertation, Stanford University, Stanford, California.

- [52] Konolige, K. *Experimental Robot Psychology*. Technical Note 363, SRI Artificial Intelligence Center, Menlo Park, California, 1985.
- [53] Konolige, K. (1986). Resolution methods for quantified modal logics. Forthcoming Artificial Intelligence Center Tech Note, SRI International, Menlo Park, California.
- [54] Konolige, K. On the relation between default logic and autoepistemic theories. 1987. submitted to IJCAI87.
- [55] Konolige, K. and Myers, K. Representing defaults with epistemic concepts. 1987. submitted to AAAI87.
- [56] Kripke, S. A. (1959). A Completeness Theorem in Modal Logic. *Journal of Symbolic Logic* 24, pp. 1-14.
- [57] Kuo, V. (1984). A formal natural deduction system about knowledge: modal logic W-JS. Unpublished manuscript, Stanford University.
- [58] Ladkin, P. "Comments on the Representation of Time," in *Proceedings of 1985 Workshop on Distributed Artificial Intelligence*, Sea Ranch, California, pp. 137-158 (1985).
- [59] Lamport, L. "Times, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM* Vol. 21, No. 7, pp. 558-565 (July 1978).
- [60] Langlotz, C. Siglunch talk. February, 1987. Stanford University.
- [61] Lansky, A. L., "A Representation of Parallel Activity Based on Events, Structure, and Causality," *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon (1987).
- [62] Lansky, A.L. "GEMPLAN: Event-based Planning Through Temporal Logic Constraint Satisfaction," Forthcoming Working Paper, Artificial Intelligence Center, SRI International, Menlo Park, California (1987).
- [63] Lansky, A.L. "Behavioral Specification and Planning for Multiagent Domains," Technical Note 360, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).
- [64] Lansky, A.L. "A 'Behavioral' Approach to Multiagent Domains," in *Proceedings of 1985 Workshop on Distributed Artificial Intelligence*, Sea Ranch, California, pp. 159-183 (1985).
- [65] Lansky, A.L. "Specification and Analysis of Concurrency," Ph.D. Thesis, Technical Report STAN-CS-83-993, Department of Computer Science, Stanford University. Stanford, California (December 1983).
- [66] Lansky, A.L. and S.S.Owicki, "GEM: A Tool for Concurrency Specification and Verification," *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pp.198-212 (August 1983).

- [67] Levesque, H. J. (1982). A Formal Treatment of Incomplete Knowledge Bases. FLAIR Technical Report No. 614, Fairchild Laboratories, Palo Alto, California.
- [68] Levesque, H. J. (1984). A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence*, Houston, Texas, pp. 198-202.
- [69] Lifschitz, V. "Circumscription in the Blocks World," Computer Science Working Memo, Stanford University, Stanford, California (1985).
- [70] Lifschitz, V. "On the Semantics of STRIPS," *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon (1987).
- [71] Litman, D. *Plan Recognition and Discourse Analysis*. PhD thesis, University of Rochester, Rochester, New York, 1985.
- [72] Lukasiewicz, W. Two results on default logic. In *Proceedings of the American Association of Artificial Intelligence*, pages 459-461, University of California at Los Angeles, 1985.
- [73] Manna, Z., and Waldinger, R. W., "A Theory of Plans," *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon (1987).
- [74] Manna, Z. and P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications," *ACM Transactions on Programming Languages and Systems*, 6 (1), pp.68-93 (January 1984).
- [75] McCarthy, J. Applications of circumscription to formalize common sense knowledge. In *Proceedings of the Workshop on Non-Monotonic Reasoning*, American Association for Artificial Intelligence, Menlo Park, California, 1984.
- [76] McCarthy, J. Circumscription — a form of nonmonotonic reasoning. *Artificial Intelligence*, 13(1-2), 1980.
- [77] McCarthy, J. *et. al.* (1978). On the model theory of knowledge. Memo AIM-312. Stanford University, Stanford.
- [78] McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of Artificial Intelligence. In *Machine Intelligence 4*, B. Meltzer and D. Michie editors, Edinburgh University Press, Edinburgh, Scotland, pp. 120-147.
- [79] McDermott, D. and Doyle, J. Non-monotonic logic I. *Artificial Intelligence*, 13(1-2):41-72, 1980.
- [80] McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans." *Cognitive Science*, 6, pp. 101-155 (1982).
- [81] Milner, R., *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer Verlag, New York (1980).

- [82] Minsky, M. A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.
- [83] Moore, R. C. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1), 1985.
- [84] Moore, R. C. (1980). Reasoning about knowledge and action. Artificial Intelligence Center Technical Note 191, SRI International, Menlo Park, California.
- [85] Nilsson, N. Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, California (1980).
- [86] N. J. Nilsson. *Shakey the Robot*. Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, California, 1984.
- [87] N. J. Nilsson. *Triangle Tables: A Proposal for a Robot Programming Language*. Technical Note 347, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.
- [88] Owicki, S. and L.Lamport, "Proving Liveness Properties of Concurrent Programs," *ACM TOPLAS* 4, 3, pp.455-492 (July 1982).
- [89] Pednault, E. P. D., "Toward a Mathematical Theory of Plan Synthesis," Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, California (1986).
- [90] Pednault, E. P. D., "Solving Multiagent Dynamic World Problems in the Classical Planning Framework," *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon (1987).
- [91] Pelavin, R., "A Formal Logic for Planning with a Partial Description of the Future," Ph.D. Thesis, Department of Computer Science, University of Rochester, Rochester, New York (1986).
- [92] Pollack, M. E. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the Workshop on Planning and Reasoning about Action*, pages 217-233, Timberline, Oregon, 1986.
- [93] Pollock, J. L. *Knowledge and Justification*. Princeton University Press, Princeton, New Jersey, 1975.
- [94] Reiter, R. A logic for default reasoning. *Artificial Intelligence*, 13(1-2), 1980.
- [95] Reiter, R. and Criscuolo, G. Some representational issues in default reasoning. In Cercone, N. J., editor, *Computational Linguistics*, pages 15-27. Pergamon Press, Elmsford, New York, 1983.
- [96] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.* 12, pp. 23-41.

- [97] Robinson, J. A. *Logic: Form and Function*. Elsevier North Holland, New York, 1979.
- [98] Rosenschein, J. S., and Genesereth, M. R. (1984). Communication and cooperation. Heuristic Programming Project Report 84-5, Stanford University, Stanford, California.
- [99] Rosenschein, S. J., "Plan Synthesis: A Logical Perspective," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, British Columbia (1981).
- [100] Sacerdoti, E.D., *A Structure for Plans and Behavior*, Elsevier, North Holland Publishing Company, New York, New York (1977).
- [101] Sato, M. (1976). *A study of Kripke-type models for some modal logics by Gentzen's sequential method*. Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan.
- [102] Shoham, Y. and Dean, T., "Temporal Notation and Causal Terminology," Working Paper, Department of Computer Science, Yale University, New Haven, Connecticut (1985).
- [103] Shoham, Y. "Chronological Ignorance: Time, Nonmonotonicity, Necessity and Causal Theories," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania (1986).
- [104] Shoham, Y. "What is the Frame Problem," *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon (1987).
- [105] Shoham, Y. "A Logic of Events," Working Paper, Department of Computer Science, Yale University, New Haven, Connecticut (1985).
- [106] Shoham, Y. and T.Dean, "Temporal Notation and Causal Terminology," Working Paper, Department of Computer Science, Yale University, New Haven, Connecticut (1985).
- [107] Smullyan, R. M. (1971). *First-order logic*. Springer-Verlag, New York.
- [108] Stalnaker, R. C. A note on nonmonotonic modal logic. 1980. Department of Philosophy, Cornell University.
- [109] Stickel, M. E. (1982). A nonclausal connection-graph resolution theorem-proving program. *Proceedings of the AAAI-82 National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, pp. 229-233.
- [110] Stickel, M. E. (1985). Automated deduction by theory resolution. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California.

- [111] Stuart, C. J., "An Implementation of a Multi-Agent Plan Synchronizer Using a Temporal Logic Theorem Prover," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California (1985).
- [112] Tate, A. "Generating Project Networks," *IJCAI-77, Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, pp. 888-893 (August 1977).
- [113] Touretzky, D. S. *The Mathematics of Inheritance Systems*. Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.
- [114] S. Vere. Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246-267, 1983.
- [115] Weyhrauch, R. (1980). Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence* 13, no. 1-2.
- [116] Wilkins, D. E., "Domain-Independent Planning: Representation and Plan Generation," *Artificial Intelligence*, 22, pp. 269-302 (1984).

END

8-87

DTIC