Planning with Hypothetical Reasoning

by

James O. Pendergraft

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

APPROVED:

[illegible]
John W. Roach, Chairman

[illegible]
Jane T. Nutter

[illegible]
Daniel M. Kervick

August, 1988

Blacksburg, Virginia

Planning with Hypothetical Reasoning

by

James O. Pendergraft

Dr. John Roach, Chairman
Computer Science

(ABSTRACT)

A planner driven by a causal theory and based on hypothetical reasoning is constructed and discussed. The task is approached from the fundamentals of time and event logics, and causality, resulting in a planner suitable for modeling a wide variety of realistic problem domains, and capable of reasoning in an intuitive manner about dynamic domains. The underlying causal theory drives the planning process directly and, in conjunction with the uniform representation of time and causal facts, allows elegant solutions to planning problems. A new type of planning problem, the indirect goal problem, is identified and solved. It is also shown that previous planners cannot solve this type of problem. The frame problem is discussed in detail, and given a computational definition, suitable for allowing objective comparison between different approaches. The hypothetical reasoning approach is shown to allow an elegant solution to the frame problem appropriate for planning systems.

Acknowledgements

I owe a great deal of credit for the success I have had to the support and encouragement I have received from my parents, the rest of my family, and many excellent teachers from my first years in school through high school and at Virginia Tech. Thanks to all of you, especially Mom and Dad.

I would also like to thank my committee members, particularly Dr. Roach, my chairman, who has probably worked on this as hard as I have. Thanks also to Texas Instruments and the Center for Innovative Technology for gifts and grants that helped pay for the research. I also want to thank and particularly (also graduate students at Virginia Tech) for help on parts of this thesis.

Last, and most important, thanks to my wife for her love, encouragement, and patience.

Table of Contents

1. Introduction

Hypothetical reasoning, a topic related to consistency maintenance, counterfactual logic, and truth maintenance, has enjoyed a recent popularity in AI as a tool for solving a variety of problems, such as knowledge base maintenance and perception. A combination of Recent progress at Virginia Tech in counterfactuals, and other work in multiple agent robotics simulation, made the idea of a consistency based planning system attractive for both intuitive and practical reasons. The simulations already implemented used consistency maintenance for robot perception to recognize consequences of changes in knowledge triggered by perceived facts. Consistency maintenance worked very well in that application and extending it to cover other aspects of robot behavior seemed possible. Other planners had been implemented and used by the simulated robots, but they were not satisfying because they were very limited. Since they were based on GPS, their planning capabilities were too simple to represent multiple agents or introspection, both important needs for other ongoing work in plan recognition for competing and/or cooperating agents.

Maintaining the consistency of robot beliefs is one research application of the hypothetical reasoning engine we use at Virginia Tech. Hypothetical reasoning is a process in which the expected consequences of a change are calculated. For example, 'What if the robot moved to the other room?' poses a hypothetical whose consequences could be of practical interest. In answering, we want to figure out that the robot is no longer in the room from which it came (because it would be inconsistent for the robot to be in two rooms at the same time), and that the things the robot was carrying, if any, also moved (because anything that is carried is in the same place as the agent carrying it).

Intuitively, consistency maintenance is desirable because we believe that it is similar to how human beings think. It is not the expected that gets our attention, it is the unexpected. Consistency maintenance excels at recognizing the unexpected, as exemplified by its use for robot perception. The idea in planning is to find possible courses of action to accomplish a goal; consistency maintenance helps by throwing out impossible ideas, channelling progress in useful directions. It channels by eliminating impossible (inconsistent) paths, and by limiting work to those paths that are relevant in some way to the goal. We consider its operation similar to our human associative memory, which makes consistency maintenance a very promising basis for further work in emulating human intellectual capabilities.

From one viewpoint, planning is a closed issue; provably complete and correct planners have been implemented. Practically speaking, however, previous problem solving systems are not useful for planning actions in a complex, changing environment. They are too inefficient, because they are complete. A complete system must consider far too many possibilities so that it can always find a solution where one exists. A new approach is needed, one that can 'think' in a more useful manner. Any complete, correct planner will find real world problems intractable. There is too much information, and too many unknowns. Consistency maintenance is one new direction in which to look for an approach that can process facts in a new manner to come up with useful solutions. Its possible worlds paradigm is natural for representing uncertainty and contingency. Consistency maintenance is also attractive because it is so well suited to several other 'mental' tasks, such as perception, and maintenance of a knowledge base. It may be possible to use consistency maintenance as a basis to unify many AI tasks.

This background provided the motivation for our research. We were interested in exploring various planning paradigms, and developing one for use with consistency maintenance. This alone, however, was not sufficient. It is important to be able to show a strong formal basis for any new work. This work is very closely tied to established fields of tense logic and modal logic so that questions about the system can be answered in a formal way. The logical systems also bring a well known semantic model to the new work.

The result of our research is a consistency based planning paradigm equivalent to particular modal and tense logical systems, and an implemented planner. This thesis only deals with the problem of constructing an initial plan based on the agent's beliefs; we have not attempted to handle plan repair or execution, although these topics are mentioned in the future research section. Stemming from early work by Roach [Roach, 1972], we have also identified a new type of planning problem that no earlier system (of which we are aware) is capable of solving. We call it the Indirect Goal Problem. It is characterized by a goal that is not a direct result of any action specified within the system. In systems such as STRIPS, where there are no indirect consequences of an action, indirect goal problems cannot even be formulated. For previous planners in which indirect consequences of actions are possible, the definition of new operators to solve the goal directly is necessary, thus eliminating the indirect aspect of the goal. Our system can solve this type of problem, in addition to nonlinear problems and problems such as the Yale Shooting Problem [Hanks and McDermott, 1987].

This thesis also includes an extensive discussion of the frame problem. The problem is defined in a new way that allows it to be tied very closely to computational issues, so that an objective comparison of various approaches can be made. We have also defined a new subproblem of the frame problem that we call the persistence problem. Each of the aspects of the frame problem are discussed in detail, and we show how our system handles each aspect.

We have optimized some aspects of the final implemented system at the expense of others. In particular, there is a necessary tradeoff between time, space, and the difficulty of implementing a system for a particular domain. Ease of use has been our first concern, in order to improve the reliability and usefulness of the methodology. Next we emphasized the speed of the system since that seems more of an issue today than storage space. Within these constraints, storage space was conserved wherever possible.

The mechanism of our planning system is a counterfactual reasoning engine, as we have mentioned. In order to use the engine, we must describe the world in terms of rules and facts that the engine can understand. In describing the world, we have not attempted to describe how one makes a plan. Instead, we have described the world with an event and causality logic and other facts about the world. It is this logic that drives the system. Our system has no 'planning algorithm' as other planners have had. The engine simply finds the possible worlds that are consistent with our description (our rules plus the initial state and goals), and since we provide consistency axioms for causality, the resulting worlds all contain causal chains resulting in the goals.

We think that systems such as the one we have developed, and describe here, will be the basis for the next generation of planning systems. Consistency maintenance and hypothetical reasoning are the logical successors to the previous nonmonotonic approaches like default reasoning and circumscription. We have found hypothetical reasoning to be a powerful tool for solving realistic problems because it is versatile, intuitive, and avoids some problems posed by other nonmonotonic approaches. We have managed to incorporate the best of several paradigms, while avoiding some of the pitfalls. Although we have not created the perfect problem solving system, we believe that we have made a step in the right direction, and created both a basis for future work and a methodology to compare and contrast past and future work on the frame problem. The generality of our solution makes it an excellent starting point for future advances. Our generality springs from two main sources. First, there is no explicit 'planning algorithm'. The system works from its knowledge of the world and the goals. Second, our temporal logic is very basic, with simple constructs from which more complex concepts can be built. In addition, our future research section presents several ideas for improving the system's practical and theoretical usefulness.

This thesis has grown out of previous work in four primary areas: planning, the frame problem, counterfactual logic, and temporal/event logics. Instead of a traditional chapter containing a literature review, the previous work in each of these areas is discussed at the beginning of the chapter on that topic.

2. Problem Analysis

Within artificial intelligence, there are several major fronts on which research is active. Among these areas are knowledge representation , natural language understanding, theorem proving, vision and pattern recognition, and problem solving and planning. In this chapter we take a first look at the planning and knowledge representation issues discussed in this thesis.

Often a major hurdle that must be crossed before 'solving' a problem in computer science is the task of defining the problem in a manner amenable to testing so that various potential solutions can be quantitatively compared. A peculiarity of AI in particular is that a 'correct' solution is not always the most desirable. The problems that we want to solve with AI techniques are often computationally intractable, yet researchers either tend to focus on sound, complete solutions or just try to get the job done, so to speak, without any formal basis that allows their work to be extended and improved in a logical, consistent manner. We would like to unify these two approaches by creating a system that is well understood theoretically, yet still useful for practical problem solving applications.

2.1. Problem-Solving Systems

Problem solving is an area with a long history in the AI field. The technology has evolved considerably from problem solving applications of the first simple theorem provers to systems such as we have developed. Our research has concentrated on the knowledge representation and planning areas. Both of these are very large topics with many problems remaining to be solved; some recognized problems are still not even well defined.

Knowledge representation is a topic that forms the basis of nearly all AI research. A general solution is still nowhere in sight, although research is actively progressing towards that goal (for a recent example of this sort of work, see [Barwise and Perry, 1983]). We are primarily concerned with the specific issues concerning problem solving systems, particularly time, change, and causality. These issues are at the heart of problem solving, and must be satisfactorily answered to proceed further.

There is active research in the area of improved planning algorithms, and the rigorous description of these algorithms. There is also research in the various aspects of the frame problem, and on various nonmonotonic logics that might be useful. Some other planning problems that are currently being explored include resource reasoning, plan recognition, cooperating and competing agents, and how to handle incomplete and/or incorrect knowledge.

One major focus for recent research into problem solving systems has been the development of methods using nonmonotonic logics to support reasonable inferences, as opposed to logically valid inferences. This is of interest because humans certainly are not limited to only logically valid thought processes. As yet, however, there is no satisfactory algorithm or theory for making reasonable inferences.

2.2. Specific Area of Exploration

We have approached this work with the idea of finding out how to use hypothetical reasoning to best advantage on as broad a range of outstanding planning problems as possible, rather than picking one and trying to solve only that one. This research has led up many blind alleys, but we have identified several problem areas for which hypothetical reasoning is an appropriate method of solution.

We wanted a state-of-the-art planner, capable of handling nonlinear problems, and other difficult problems such as the Yale Shooting Problem [Hanks and McDermott, 1987]. Further, we wanted to re-examine the basis of planning, and start from first principles: event and causal logics. To approach this goal, we created a simple, elegant temporal and causal model, and used it to drive a hypothetical reasoning engine, thereby forming a planner. We use the hypothetical reasoning engine WIF (standing for 'What IF') [Roach, Eichelman, and Whitehead, 1985]. In the process of implementation, several interesting questions were raised, having to do primarily with modeling time and change, and with the frame problem. Further work on these led to improvements in the planning system.

The areas that have proved tractable to us using hypothetical reasoning methods are the following:

- modeling time in a manner useful for problem solving in real-world domains
- a formal definition of our time model
- modeling causality and change in the same types of domains
- a planning algorithm based on our hypothetical reasoning engine, WIF, using these methods, capable of solving previously identified types of planning problems and the indirect goal problem.
- a definition for the frame problem in computational terms
- a new solution to the frame problem

We have also included several example domains, corresponding to important planning problems that we are interested in solving. Also included are some guidelines for setting up similar systems for other domains. Our research has raised several new questions, some of which we have answered, and some of which are discussed in Chapter 8 in the section on further research.

3. Hypothetical Reasoning

Since hypothetical reasoning is the inspiration and the basis for this work, we will start by discussing the topic in general, and also as computationally implemented here at Virginia Tech. We will start with a review of the logical foundations of the topic, and mention some other uses that have been found for it. Our hypothetical reasoning engine (WIF) was developed by Roach and Eichelman, and implemented by Whitehead in [Roach, Eichelman, Whitehead, 1985] and has been applied to causal modeling and intuitive physics by Whitehead [Whitehead, 1988]. These references give a complete account of the theoretical and implementation details of WIF. After the literature review, we begin with an overview of the system from a theoretical perspective. The next section describes WIF from a user's viewpoint. Following that is a discussion of the practical aspects of its use.

3.1. Literature Review

Counterfactual logic, hypothetical reasoning, and consistency maintenance are all closely related aspects of the same concept that have received a lot of attention recently in AI as a possible solution to problems related to knowledge representation, change, and consistency. This is a brief review of the ideas on which this thesis is based.

A digression into terminology is in order now, because some confusion exists in the field and we want to avoid that in our discussion. A counterfactual is a statement in the form of a conditional, but with a false antecedent. A hypothetical is a statement assumed true, even though it may actually be true or false. For example, 'What if Kennedy had not been elected?' poses the hypothetical 'Kennedy was not elected', while the sentence 'If Kennedy had not been elected president, then he would not have been assassinated' is a counterfactual. Whether this counterfactual is true or false is an open question, depending on which view of counterfactuals is taken. Some authors such as J. L. Mackie [Kvart, 1981] deny that counterfactuals are statements, and claim that therefore they do not even have truth values. We and all the authors reviewed here do believe that it has a truth value, although the method determining the truth value of any particular counterfactual is hotly debated.

In David Lewis's [Lewis, 1974] work, he supposes that a counterfactual is true if the conclusion is true in all of the closest possible worlds consistent with the antecedent. The exact definition of 'closest possible worlds' is context dependent in his theory, and requires an ordering on the importance of facts, with less believed facts being the first to be thrown out in the attempt to find a consistent resolution to the contradiction.

Nicholas Rescher [Rescher, 1964], in contrast to Lewis's logic-based system for counterfactuals, presents an algorithm for producing all maximal consistent subsets of the original world that can be unioned with a set of hypotheticals. A counterfactual, $p \supset q$, is true in Rescher's system if and only if q is in all of the maximal consistent subsets of the world produced by the algorithm with $\{p\}$ as the set of hypotheticals. He requires a partial ordering on the believability of facts, so that some facts are more 'believable' than others. Using this ordering, a belief ordering between the various worlds produced by the algorithm can be produced, any optimum of which he calls a preferred maximal mutually-compatible subset, or PMMC, which may not be unique. Because consistency is not a decidable or even semidecidable in first order logic, some restrictions on the power of the language were assumed in order to make the problem tractable.

This ranking of beliefs by a believability rating characterizes the epistemic school of counterfactual logic, as opposed to the metaphysical school of counterfactuals. In the epistemic school, one can discuss the degree of truth of a counterfactual, as opposed to whether or not one is valid in a metaphysical system such as Lewis'. For realistic problems we feel that the range of expressivity possible with the epistemic school is more appropriate.

Building on Rescher's work, a new algorithm to reduce the computational effort involved has been developed at Virginia Tech [Roach, Eichelman, and Whitehead,1985]. This is the basis for the hypothetical reasoning engine described in the next section.

3.2. The Hypothetical Reasoning Engine

A 'hypothetical reasoning engine' is a system that produces the consequences of a hypothesis given a description of the world and a set of constraints on valid states of the world. Following in the vein of Rescher, but using the algorithm from [Roach, Eichelman, and Whitehead, 1985], the hypothetical reasoning engine WIF has been implemented in PROLOG on a VAX system. It also now runs on a Texas Instruments Explorer II system.

WIF has been used for several projects in addition to this one. One previous use was for adding perceptions to a knowledge base in a robot planning/simulation system [Graham, 1986]. This system employs a simple STRIPS type planner. When new information is added to the robot's beliefs, WIF makes sure that it is consistent with previous knowledge. If not, then it has found a case where beliefs have changed. For instance, let's assume that I saw the broom in the closet, and that I assume that things stay where I last saw them unless I know otherwise. I will believe that the broom remains in the closet until I see it elsewhere (or see that it is not in the closet), at which point WIF will recognize the contradiction, and throw out the belief that the broom is still in the closet.

A major accomplishment using the system was the development of HOIST, an expert system based on qualitative physics [Whitehead and Roach, 1987]. No expert was used to create this fault diagnosis expert system for a large mechanical system. Instead, a causal model based on intuitive physics was built from the actual blueprints of the machine; this model was used by WIF. Fault diagnosis was accomplished by hypothesizing a symptom of a malfunction in the machine; then WIF would restore consistency by creating various possible worlds in which the parts of the mechanism malfunction. The paper reporting this project won the Grand Prize in the Texas Instruments AI in Industrial Automation Contest in July, 1987.

3.3. The Syntax and Corresponding Semantics

The user of WIF starts by supplying the system with facts about the state of the world and consistency rules describing consistent and inconsistent relationships between facts. WIF assumes that the initial facts are consistent under propositional logic, i.e. there is no direct contradiction in the facts or implied by the rules and facts taken together. Then, when a hypothesis is supplied to WIF, it returns the set of all consistent worlds created by the process described in this section.

WIF includes the notion of a 'believability index', called a modal category, which is associated with each fact. A modal category of 0 (or less) is associated with an irrefutable fact. In general, a higher number denotes a fact that is less certain.

Facts in WIF are represented in the following format:

```
(fact (predicate_name arguments) n)
```

where *n* is the modal category, a nonnegative integer, and *arguments* is one or more literal atoms used as arguments for that predicate. Negated facts are stored as:

```
(fact (not (predicate_name arguments)) n)
```

A consistency rule is written as an implication with preconditions on the left hand side (LHS) and consequences on the right hand side (RHS). The general format is as follows:

```
((antecedent_1 arguments)
 (antecedent_2 arguments)
 ...
 (antecedent_n arguments) -> (consequent_1 arguments)
  •                          (consequent_2 arguments)
                             ...
                             (consequent_m arguments))
```

where the antecedents and consequents are predicate names. Unlike facts, the arguments of the predicates may contain variables[1]. Since rules are applied to facts with pattern matching followed by instantiation, this is necessary to make the rules general.

Given a set of facts (an initial world), a set of rules, and a set of hypotheses, the consistent possible worlds are derived by the following algorithm. For each hypothesis, attempt to match the hypothesis to a predicate and its arguments on the LHS of each consistency rule. For any that match, check whether the rest of the LHS is believed. If so, add the RHS to the set of hypotheses. If this produces a contradiction with the world[2], create n-1 hypothesis sets, where n is the number of antecedents in the rule, where each set is the old set plus the negation of one of the n-1 other antecedents. Throw out any sets containing a direct contradiction.[3] If no sets remain, or if n = 1, then fail for this hypothesis set: there is no possible resolution to the inconsistency. Apply the same process recursively to each resulting hypothesis set, until no further rules can be applied. Each

[1] It is now possible to use variables in facts as well as rules, although since we do not make use of the ability, we have not described this improvement here.

[2] A contradiction is found if the negation of the hypothesis is in the initial world.

[3] A direct contradiction is found if the negation is in the hypothesis set.

remaining set is a possible world in which the hypotheses are true, if none remain, the hypotheses are inconsistent with the original world.

Note that there is no mention of the use of the modal categories in this description of the algorithm. This is a simplification of the actual algorithm that we have made in order to present it in a more understandable manner. In terms of the algorithm given, the modal category of a fact can make it behave as if it were in the hypothesis set for purposes of finding contradiction and such, but the fact is not tested against the LHS of any rules in order to make further inferences. Our planning system does not use the modal category feature at all, so the algorithm as originally stated is correct for our system. In several places throughout this thesis, we mention possible applications of modal categories, but they are not in the current system.

Let us now turn to an example of WIF in operation. Consider a situation in which there is a robot, `Robot1`, two rooms, RoomA and `RoomB`, and a mop, `Mop1`. `Robot1` starts in `RoomA` holding `Mop1`. The facts expressing the system's initial knowledge are these:

```
(robot Robot1)
(room RoomA)
(room RoomB)
(mop Mop1)
(in RoomA Robot1)
(in RoomA Mop1)
(holding Robot1 Mop1)
```

We now need to give the system some rules about the world. Some useful rules in this sort of world are that carried things stay with the carrier and that nothing can be in two places at once. A leading asterisk (`*`) signifies a variable in these rules. The are expressed like this:

```
(in *room1 *x)(in *room2 *x) -> (= *room1 *room2)
(in *room *agent)(holding *agent *thing)
```

```
                    -> (in *room *thing)
```

Now let us suppose that the robot has moved to `RoomB` and is now there. If we supply `(in RoomB Robot1)` as a hypothesis, this is what happens. Taking the only hypothesis it has, it looks up all occurrences of the `in` predicate on the LHS of rules. Both rules fit this criterion. Instantiating the first rule, it needs to find whether `(in *room2 Robot1)` is true. In fact, two instances of this form are true. The first is the new hypothesis `(in RoomB Robot1)`. From this WIF infers `(= RoomB RoomB)`, which is fine. The second case is the fact about the starting location of the robot `(in RoomA Robot1)`. From this, WIF infers that either `(= RoomA RoomB)` or `(not (in RoomA Robot1))`. Since equality is determined by pattern matching[1], the first solution fails, so `(not (in RoomA Robot1))` is added to the hypothesis list. Instantiating the second rule with the original hypothesis, it needs to find out whether `(holding Robot1 *thing)` to see if the rule applies. In fact, `(holding Robot1 Mop1)` is found, so it concludes that either `(not (holding Robot1 Mop1))` or `(in RoomB Mop1)`. In the first case, no new matches are generated since `(not (holding * *))` does not appear on any LHS. In the second case, `(in RoomB Mop1)`, there is a match with the first rule, resulting in `(not (in RoomA Mop1))`, as also occurred when the robot moved.

Therefore we have two consistent possible worlds for which the hypothesis, `(in RoomB Robot1)`, is true. Note that the addition of the negation of a fact entails the deletion of that fact if it exists. The first answer is:

```
(in RoomB Robot1)
```

[1] For constants in our system, equality reduces to token identity. This means that nothing can have two names. This can be a problem in some applications, but it can be overcome simply by keeping track of synonyms, and redefining the equality predicate in PROLOG to check this list, instead of trying to blindly unify its arguments.

```
(not (in RoomA Robot1))
(not (holding Robot1 Mop1))
```

while the second is:

```
(in RoomB Robot1)
(not (in RoomA Robot1))
(in RoomB Mop1)
(not (in RoomA Mop1))
```

These are the two possible results of introducing the hypothesis. There are not any others within the scope of knowledge of the system. Hopefully this example has made the operation and use of WIF clearer to the reader. More detailed information on WIF and examples of its use can be found in [Graham, 1986], [Whitehead and Roach, 1987], and[Whitehead, 1988], as well as later in this thesis.

3.4. Practical Considerations

WIF is a very useful tool for consistency maintenance applications. However, it does have several drawbacks. It is not particularly fast, at least in the HC PROLOG interpreter running on our microVAX. This is to be expected, given the objective of producing all possible consistent worlds from the original world. There is a speed improvement on the TI Explorer II by a factor of approximately 5.

There may be some concern about the combinatorial explosion resulting from the algorithm used by WIF. This will be discussed in detail in section 7.3, but we will simply say now that the search is limited by the number of relevant facts in the world. If everything in the world is related to the hypothesis, then the whole world will be within the scope of the search. Fortunately, the effects of changes in the real world are very localized in the vast majority of cases[1], so this explosion does not actually occur. We leave the cases where it does explode to future researchers (with much faster computers).

The application of modus tollens to rules can be selectively suppressed on a rule by rule basis, although it can be explicitly stated as the contrapositive of each rule[2]. It cannot be implemented in the general sense because of the restrictions on the expressiveness of the system that allow consistency to be decidable. The version applied is not a true modus tollens for this reason, but it prevents the creation of inconsistent worlds in which modus tollens can derive the inconsistency. For the same reason, a true existential quantification cannot be implemented, although limited forms suitable for most applications are in place.[3] It is obvious that since we have a universal quantifier and no existential, we do not have a true negation either. That does not limit us because we do not really need it ; there are no variables in our facts. Rules of the form $\forall x(\sim P(x) \supset P(x))$ can be used to simulate facts of the form $\forall x\ P(x)$, if necessary, so some variables can be used in 'facts'. Our negation is inherited from WIF, not from PROLOG, so it is not a simple negation by failure. For predicates of lower modal category than the threshold specified in the call to WIF, negation is by failure (as mentioned previously, we do not use these). For those equal to or higher than the threshold, failure allows the system to create two possible worlds, one in which the predicate is true and one in which it is false.

[1] This is a useful assumption for a planning system, even though it certainly has not been proven, and things such as chaos theory suggest that it is false. Remember that we are trying to create a useful planner, whose model need not be perfect, but just good enough to get by. Our model for planning certainly doesn't always figure out every ramification of an event, but it works enough to be useful.

[2] See section 7.3.1 for a discussion of modus tollens in WIF and how it affects our system.

[3] WIF cannot perform true existential quantification because it cannot create new individuals. In order to accomplish this, one has to go outside of WIF. An example of this can be seen later where we do this to create new time points during the planning process.

As with any depth-first algorithm of this type, there exists a potential for an infinite left-recursive loop. We have avoided this in our rules, but domain rule writers must also be careful of this difficulty.

These shortcomings aside, WIF has several very useful features and applications. Since the rules are indexed by the predicates they contain in the LHS, lookup of rules to check for a given hypothesis is very fast; no search is involved. A very useful technique for building up the initial set of facts for WIF is to start with an empty world and add primary facts; their secondary consequences and interactions are then derived automatically[1].

The order of the rules is not significant to the final result that WIF produces. It can make a very large difference, however, in how long it takes to find the answer. By ordering the rules to rule out likely inconsistencies before rules likely to generate more possible worlds, the search depth at which contradictions are found can be cut dramatically, resulting in a corresponding time savings. Proper ordering of the rules can also help avoid left-recursion.

We do not state the modal categories of the facts used in the examples in this thesis. We have also omitted the `fact` header on each fact and the outer parentheses from each rule. None of these are significant to the content of this thesis. The use of modal categories for facts is a complex topic. The application of modal categories is discussed in footnotes to the text, and in the future research section.

[1] A primary fact cannot be derived from other facts, where a secondary fact can. For instance, in BLOCKS world, OVER predicates are secondary facts derived from ON predicates.

4. Time and Change

In this chapter we discuss several related issues of fundamental importance to problem solving in realistic domains. In particular, this chapter discusses the difficulties of modeling an environment changing over time. By a realistic domain, we mean primarily physical action in a world approximately as dynamic and complex as own world. There are other types of domains, such as electronic circuit design, that are also important areas for research, but the real challenge in this field seems to lie in the sort of complex domains that people understand so well yet seem so computationally intractable. Ideally of course, all domains are part of the 'reality' domain, so the examples given above are not exceptions, just very difficult.

Several problems immediately arise as both difficult and important to solve in order to model these sorts of domains satisfactorily. The first and biggest is the question of knowledge representation. Although it is very interesting, the general knowledge representation problem is beyond the scope of this thesis. Instead, we have concentrated on a particular aspect of this problem: the representation of time, change, and causality within a domain. For other aspects of the world, we use first order logic statements as representations of specific facts about the modeled domain.

Even concentrating just on the issues of time and change, there is a great deal of previous work to examine. We start by examining some of the many earlier approaches to these modeling difficulties, from both philosophical logic and computer science perspectives. As we proceed to discuss our methodology, comparisons to these earlier methods will be made when appropriate.

The frame problem is rooted in the difficulty of representing a changing world, the topic of this chapter. Chapter 7 discusses the frame problem in detail, and then explains how our system solves it. The solution we present is based on the representations introduced in this chapter.

4.1. Time/Event Logic Literature Review

The nature of time has been discussed and debated since the time of the very earliest thinkers. There are many viewpoints on the matter, each with a corresponding theory. Some theories are limited to a particular type of model for time; others are adaptable to any model for time, within certain basic assumptions. The latter are used and discussed here, as one objective of this thesis is to describe a model of time useful for problem solving systems. The 'real' nature of time is not important except insofar as it determines how a planning agent, such as a person, represents time.

There are two major types of temporal logics; those that take time points to be primary, and those that take time intervals to be primary. Since they are interdefinable [Van Benthem, 1983], the choice of one over the other is primarily a decision of convenience. Concepts that relate intervals to each other are more concise in an interval based logic, while relations between points are easier to describe in a point based logic. The first temporal logics were based on the predicate calculus. They simply added an extra argument to each predicate to represent the time for which the statement was true. These times could be variables so they could be existentially and universally quantified. In the interest of conciseness and understandability, and later in order to increase the expressive power of the logical representation, entirely new logics were developed just to handle temporal constructions.

A. N. Prior developed a major class of temporal logics called tense logics which take time points as primary. In [Prior, 1967a] and [Prior, 1967b] he defines and discusses the basic tense logic $\mathbf{K}_t$, and many extensions and variations of it. The variants arise from changes in the assumptions made about time. These choices are whether time is linear, branching, circular, or acyclical, whether it is unbounded in one or both directions, and whether time instants can be mapped to a discrete, dense, or continuous model. More recently, Humberstone has developed an interval based semantics for tense logics [Humberstone, 1979].

Drew McDermott axiomatized a very ambitious temporal logic with 44 axioms [McDermott, 1982]. His work allows proofs about plans, actions, events, facts, and world histories. We, however, share Raymond Turner's doubts [Turner, 1984] about the philosophical and logical basis of the system, particularly whether so complex a system is consistent.

James Allen [Allen, 1981] has produced a similarly complex logic with which we have similar concerns about consistency [Turner, 1984]. His differs in that it takes intervals to be primary, although he does define point-like intervals that do not overlap or meet with other intervals. He also takes events as primitive. His relations are between intervals and events.

After discussing Allen and McDermott, Turner develops some properties of events and temporal relations that he thinks should be reflected by a logic on that subject [Turner, 1984]. His intuitions seem valid, although we have taken a different view of events, and many of his axioms can be proved vacuously in our system.

Working with Hayes, Allen improved upon his earlier interval based temporal logic by defining a simpler system that included and extended it [Allen and Hayes, 1987]. The simplification results from the use of only one relation between intervals in addition to the temporal precedence relation. They call this relation MEETS, and it is true whenever two intervals have a common endpoint with no overlap. This axiomatization is much more satisfying than Allen's earlier one. It is simpler and is much clearer about the relationships between time intervals, events, and points. However, we still believe that this approach will not work for planning, because for one thing, there is no way to distinguish between two events that occur at the same time. By definition, an interval is identical to everything contained within it.

Galton recently developed a temporal logic designed to handle aspect, which is a linguistic feature that appears in the grammars of some, but not all, languages. Aspect can be viewed as the difference between speaking of an event and speaking of a state: for example, the difference between 'He swam' and 'He was swimming'. While Galton's formalism is very interesting and actually uses an approach similar to ours in some ways, the extra power of his language is not needed for a planning system because anything written in one aspect can be written in the other [Galton, 1984], as demonstrated by those languages that do not employ aspect.

Pelavin and Allen [Pelavin and Allen, 1987] developed a model for concurrent actions over time. They apply it to the formulation of planning problems and gave a semantic model for the logic developed. They base their work on Stalnaker and Lewis' theories of counterfactuals and Allen and Hayes's temporal interval logics. Their approach applies actions to entire world histories instead of to states as previous systems have done. Following Stalnaker and Lewis' lead, they insist on only valid inferences. This severely limits the utility of the system because a logically valid plan works in all possible worlds.[1] Finding such a plan is highly unlikely; we are looking for plans that work in some possible world (hopefully the one closest to the real world). As far as we know, their approach has not been used to implement a planning system.

[1] For instance, if I am planning to go to the store, and I know about meteors, then in order for me to make a *valid* plan, I have to find a single plan that will work in all possible worlds, including those where a meteorite hits my car or even me.

Yoav Shoham's recent work developing a logic he calls the logic of chronological ignorance gives an algorithm for prediction in $n \log(n)$ time, but, as he says, it is not useful for planning applications. His work, or a future generalization of it, may indeed become of interest for problem solving systems. It is based on a nonmonotonic modal logic and he claims that it solves both the ramification and the qualification aspects of the frame problem. This seems very promising though it is not clear how to use it for planning applications since, as Shoham states in his book, it deals only with forward (in time) inferences.

4.2. Modeling Time

To model time for a planning system, we start with the question, "What is a *useful* model of time for a problem solving agent?". Determining the 'real' nature of time is not our goal here; we are looking for something similar to the human understanding of time appropriate for our uses.

Plans must be executed in time, and planning itself takes time. Because time is central to planning, a good understanding of time is necessary to proceed. Many models of time have been formulated, for various reasons, as discussed in the previous section. We will start by stating our assumptions about time, with a discussion about why these particular choices were made.

We have not chosen either points or intervals as primary. Later it will become clear that in our system events happen at a point and that statements or propositions are true over intervals. Points and intervals are interdefinable, and we use both for their own purposes. Our first assumptions are:

- Time points map onto the rational number line
- The ordering on time points is a total order

Many other alternatives have been discussed for these same issues, for widely varying reasons. These are appropriate alternatives to use in this problem solving system. They may or may not reflect 'reality', whatever that means, but they are enough to allow the representation of reality in a natural manner, and then reason about it.

A dense structure for time points is a clear choice for several reasons. A simpler time system, for example a non-dense linear system such as the integers, is not rich enough to model the world in general. It certainly is logically possible for an event to occur between any two others. With a structure such as the integers for time, we can eventually surpass the limit of expressiveness of the system by adding events on a finite portion of the timeline. The integers do not have an arbitrarily fine granularity, which is what we need to represent time for general purpose planning. Some would say that the solution is to use a linear timeline with a fine enough grain to represent anything that we are interested in. This is a workable solution but it has two drawbacks besides the inelegance of such a brute-force approach. First of all, it fails to capture the nature of time as we understand it, which is our goal. Second, it requires that the model of time change with the domain, a very undesirable 'feature'.

In the other direction lies a continuous timeline, such as the real numbers, or a system with an even higher cardinality. Intuitively, this seems plausible, even desirable, because nearly everything we want to model in the real world appears to be a continuous process. But what we want to model is not the structure of the processes themselves; instead we want to model the beginnings and endings of processes, or changes in processes. Actually, there is nothing in the theory that prohibits a continuous structure, but it is not necessary for anything herein. The result is that we have a timeline that is only as fine grained as we need, and no more. If a particular domain requires reasoning about continuous processes, as many realistic problems do, the necessary extension can be made without upsetting the formal basis of the rest of the system.

Notice however, that the result of planning, namely a plan, will always consist of a finite number of events (actions). Therefore, any plan could be mapped to an integer time model. The important point here is that we do not want to dictate a predetermined integer model, and therefore limit the planning process to an arbitrary granularity.

Time instants are fully ordered by the < relation. Ignoring the concerns of Einstein's Special Relativity, we intuitively understand that two things have to happen in some order; one or the other has to be first or they happen at the same time. A branching structure could be useful for considering future possibility but it is unnecessary. Because of the hypothetical reasoning basis, we are using a possible worlds paradigm. Within this model we can represent a branching timeline as a set of possible futures, each of whose time points are totally ordered. Some of the timelines (and therefore timepoints) may overlap between various possible worlds. This will form the basis for an accessibility relation in the model developed later based on the S_5 modal system. One possible world will split into several possible worlds sharing a common past when there is an uncertainty about what events occur. As a simple example, consider a plan for me to go to campus a few blocks away. I could ride my bicycle, or drive, or walk. All three of these are possibilities, reflected in the three possible worlds containing each individual alternative. A possible world, in the formal modal logic sense, is a complete timeline, in which the events (named timepoints) are fully ordered.

4.3. Facts about the World

Given a particular model of time, it is necessary to determine how facts about the state of the world are represented within that framework. Once again, knowledge representation in general is not the question, we are only interested in how time and change fit into that representation. In order to make the facts themselves as straightforward as possible, we write facts about the world as simple one or more place predicates. The question of interest in this thesis is how these facts about the world fit with the temporal aspect of the representation. Here we will state another basic assumption about the world on which we base our system.

- States are homogeneous (left-open) time intervals, over which propositions are true or false

This is not intuitively obvious and needs some explanation. Here we discuss how this assumption fits reality. By this definition, intervals during which no changes occur are homogeneous world states. In other words, an interval is homogeneous when no truth values of propositions of interest change during the interval. Which propositions are 'of interest' is up to the writer of the domain rules, based on the intended application of the planner. Just because a state is homogeneous does not mean that the state is static. There certainly can be a state in which a car is moving, whose interval is defined by the changes 'the car starts moving' and 'the car stops moving'. This example is only appropriate if propositions about the car while it is moving are not of interest. If knowledge of particular intermediate locations were needed for other reasoning, then the interval would be split into smaller intervals during which the car moved between the points of interest. This is an example of what was meant by change of 'propositions of interest' above. The domain rule writer must take care to model only those facts that are important to planning in the intended domain.

Whether static or dynamic, all states are still, by definition, homogeneous. People naturally refer to dynamic situations as states of change as in 'when the car was moving', instead of changes of state as in 'when the car was at x, x' ,x", ... '. These intervals are recognized as unified wholes even though change is occurring. This representation captures the interval as a unified whole, in spite of the change occurring within it.

The choice of left-open intervals was made by a process of elimination. If the intervals were fully-open, then there would exist a truth gap at any point where a predicate's truth value changed, since neither of two open intervals meeting at a common endpoint include the endpoint. On the other hand, closed intervals produce a contradiction at each point of change. The remaining choices are left-open and right-open intervals. Choosing right-open, and therefore left-closed, intervals would imply that the predicate had already changed truth value when the event causing the change occurred. Therefore, the left-open interval was chosen as it was the only choice remaining.

Although it will be discussed again later in section 7.5, we would like to point out now that we are not employing the Closed World Assumption (CWA). The CWA states that any fact that cannot be proven may be assumed false. This can be a useful simplification for some domains, but not for a general purpose planning system. Both true and false facts[1] are stored explicitly in our system; anything underivable is unknown.

Now that we have decided to represent facts about the world as propositions that can be explicitly true or false during temporal intervals, we can decide on a representation for a specific fact during a particular interval. Some naming conventions will be introduced here; they will be followed throughout the paper. Others will be introduced as necessary. Particular time points are represented as `t0`, `t1`, `t2`, and so on, with the implied ordering.

[1] Not all known facts are explicitly stored. In the persistence section we discuss how persistence information is derived when and if it is needed (used by the planner), instead of taking the time and space to derive and store everything ahead of time. Also, because of the nature of WIF, facts with a modal category of less than the threshold are not stored if negative. This was done for efficiency reasons by the implementers of WIF. The practical result of this is that facts like `(isa tree Reagan)` are false while `(on blocka blockb)` is unknown, even though neither are stored explicitly.

In actuality we should use rationals instead of integers but the integers will suffice for the examples herein. Any literal beginning with an asterisk (`*`) is a variable.

Base predicates (those that represent non-temporal relations) are embedded in one of three predicates containing temporal information, `start`, `state` and `stop`. There are some base predicates that express unchanging information and therefore do not need any attached temporal information. For instance, in most domains it is not necessary to keep track of just exactly when the earth orbited the sun, it can be taken for granted that this is timeless information. Anything that is unchanging within the timescope of the domain of interest can be represented similarly. This simplification is purely to make things easier and more compact, although perhaps less elegant. There is nothing wrong with timestamping this sort of information, but since it is inconsequential to the results, we sacrifice a little elegance for compactness and speed.

Each of the three predicates has a timestamp and confers some temporal information about the base predicate embedded within it. The `start` predicate means that the embedded predicate became true at the time of the timestamp. The `state` predicate says only that the predicate was true at that time. The `stop` predicate tells when the embedded predicate became false.

For example, the representation of the initial state of the classic blocks world problem is stated in the following familiar manner.

```
(state (on a b) t0)
(state (on b table1) t0)
(state (on c table1) t0)
(block a)
(block b)
(block c)
(table table1)
(state (clear a) t0)
(state (clear c) t0)
(state (clear table1) t0)
```

After the first move in solving the CAB nonlinear problem, the movement of block a to the table `table1`, the following propositions are added.

```
(start (on a table1) t1)
(stop (on a b) t1)
(start (clear b) t1)
```

Notice that we are only adding information as time progresses; nothing is changed or deleted. This is because we have not found any beliefs that we no longer believe. Our knowledge base purely additive as long as we believe what is in it. If a block moves to another location, we simply add that fact. It is only when we find out that we were wrong about where we thought it was that we have to change facts that already exist.

4.3.1. Persistence

In order to derive that facts are true at some arbitrary time (persist until that time), we need to supply a rule to allow them to be derived. The following rule achieves that. It is *not* a WIF rule. It is a backward chaining PROLOG rule, as opposed to the forward chaining WIF rules. Coding it in this manner allows us to delay deriving persistence information until it is needed, which saves both time and space[1]. Here is the rule, which simply states that a predicate is true if it `started` being true and has not `stopped` being true since[2]. The rule `highest_belief` is part of WIF and returns the highest modal

[1] The time and space savings are discussed in section 7.5.

[2] The justification for this rule can be found in section 4.5. We explicitly state the assumptions that support this rule there. In short, since we assume that change has a cause, we assume that a fact remains unchanged over any interval during which we cannot derive that it changed. This is sort of like a 'CWA for events'. In order for persistence to apply for a fact though, there must be an explicit statement about that fact in the past, so we are not using a CWA for state information, only for events. It seems futile to try and plan in a world where literally anything (consistent with the rest of the world) could happen, so we think this is a necessary assumption.

category already in the system. We add one to allow WIF to disbelieve persistence so that further actions can be hypothesized to account for the change.

```
(fact (state *fact *t) *mc) if
                              (fact (start *fact *t1) *m)
                              (not (changed *fact *t1 *t))
                              (highest_belief *high)
                              (= *mc (+ 1 *high))

(changed *fact *t1 *t) if (fact (stop *fact *t2) *)
                              (< *t1 *t2)
                              (< *t2 *t)
```

4.3.2. Domain Constraints

There is another type of world knowledge our system needs. It is the sort of everyday knowledge that we take for granted. An example is 'No two things can be in the same place at the same time'. Another is 'No one thing can be in more than one place at one time.' Using the consistency rule format discussed earlier, these rules about the behavior of objects in the physical world can be written as such:

```
(start (on *obj1 *loc) *t)
(state (on *obj2 *loc) *t)
                         ->    (= *obj1 *obj2)

(start (on *obj *loc1) *t)
(state (on *obj *loc2) *t)
                         ->    (= *loc1 *loc2)
```

These rules reflect the 'natural laws' of the modeled domain. They express relationships between facts in a consistent world state. There are more examples of these rules throughout this thesis. Notice that the rules involve one `start` and one (or more in general) `states`. This is because we catch all inconsistencies when the change causing the inconsistency occurs. Restrictions between `states` are not needed because they are protected from ever becoming inconsistent.

4.4. Events and Change

Having discussed time and the representation of facts within it, we now turn to the issue of changes. A 'change' amounts to the negation of the truth value of some predicate. In our system, a `start` or `stop` predicate represents a change.[1] The event responsible for the change takes place at the same time as the change. Another assumption about time and change follows.

- **Events are instantaneous, because they are state transitions**

The claim that events are instantaneous seems counter-intuitive, but it follows directly if events are regarded as state changes. Since a proposition is always either true or false, a state change cannot take any time and is therefore instantaneous.

An immediate objection to this might be "What about the events 'Bob picked up the book' or 'Bob changed the tire' ? They certainly aren't instantaneous". The answer is twofold. First, in our representation 'Bob picked up the book' names an interval delimited by two events 'Bob started to pick up the book' and 'Bob finished picking up the book', which is composed of many other intervals between the other events involved, such as 'Bob starting to grasp the book' and 'Bob finishing grasping the book'. Notice that we would speak of the interval as 'Bob was picking up the book' if we wanted to describe a state. The difference is in the aspectual character of the verb phrase [Galton, 1984]; one is talking about an event occurring during a time interval, and one is talking about the state during that interval. The second answer is that there is no practical difference between an instant and a nondecomposable interval [Allen and Hayes, 1987]. If the scale of the problem is appropriate, you could treat 'Bob picked up the book' as an instantaneous

[1] These look similar to the `start` and `end` predicates of Kowalski and Sergot [Kowalski and Sergot, 1985], although they were arrived at independently and serve different purposes.

event. This is true as long as no other events occur while the book is being picked up. It is clear then, that the key to using this scheme efficiently is to identify the necessary granularity for events, and treat them as nondecomposable. In our theory they are treated as instantaneous because they are nondecomposable; in an implementation they could be assigned a duration and still be nondecomposable. There are things that we do that take time, but we always follow the same routine. These are good candidates for a planner to treat as nondecomposable. For example, starting a car involves several actions but we do not plan each step: we have a single 'precomputed' procedure that is atomic *from the planner's point of view*. This simplification is safe as long as we do not have to reason about other events that occur ***during*** the starting of the car. In a single agent situation, this is fine, and even in multi-agent situations, it seems unlikely that interactions between the various processes that occur when a car is started and other external facts will be important.

Are the rising and setting of the sun events or is the spinning of the Earth just a state of change? Both are true. The spinning of the Earth triggers the sunset and sunrise events in a particular domain. Notice that we must localize the area of interest. It does us no good to spend all of our time deriving the sunsets and sunrises that occur continuously all around the world. Our planning algorithm insures that we do not compute these useless pieces of information. From this example we can see that the implementer of the rules for a particular domain must decide exactly which aspects of the world are states of change and which are events, and their interrelationships. The events should be chosen to be the inception and completion of changes important to the planning process.

An event is represented within an `occur` predicate, just as facts about the world are represented in the `start` and `stop` predicates. The `occur` statement specifies the time of the event. In general, events result in a list of state changes that `start` after the action `occurs`. Typically only agent actions are represented as events but the model is general enough to represent an event caused by any means. Since a world state is a complete description of the world and its history, any cause for an event must be in that description. We have, in essence, assumed that we know about all the possible causes for a change. This is a reasonable assumption for a planning system, because we cannot plan to perform an action of which we are unaware. In a general prediction system this would be a major limitation, but for planning it is appropriate.

Let us now examine an example of this representation. The following WIF rules are fragments of the Blocks World consistency rule set. They express the changes caused by the events pickup and putdown.

```
(occur (pickup *rob *obj) *t)
                     ->   (start (hold *rob *obj) *t)

(occur (putdown *rob *obj *loc) *t)
                     ->   (start (on *obj *loc) *t)

(start (hold *rob *obj1) *t)
(state (hold *rob *obj2) *t)
                     ->   (= *obj1 *obj2)

(start (hold *rob *obj) *t)
(state (on *obj *loc) *t)
                     ->   (stop (on *obj *loc) *t)

(start (on *obj *loc) *t)
(state (hold *rob *obj) *t)
                     ->   (stop (hold *rob *obj) *t)
```

The first two rules give the direct preconditions and results of the actions specified. The third rule states that a robot can only hold one thing. The other two say that an object cannot be held by a robot and be on something at the same time. Each of these rules is a simple, intuitive statement about the relationships between predicates. The actions have only positive direct results; facts that are no longer true are indirectly derived.

Notice how easy it would be to change or eliminate a constraint represented by one of these rules. For instance, to change the restriction that a robot can hold at most one item, only one rule needs to be changed. There is no need to change all of the rules that affect the `hold` predicate, only the one rule expressing that restriction on it, which involves only that predicate. This is a case of the principle of locality in action.

This example builds on the rules given in the previous section in that the earlier rules will not allow a block to be placed in an occupied location. Again, we see the advantage of simple, straightforward rules. We do not have to state explicitly that a location is clear before a block is placed there; other more general rules have already taken care of this.

One traditional restriction not represented here is that a block can only be picked up if nothing is on it. Adding this restriction is not difficult; we simply add a rule stating the restriction, and we need not alter previous rules. On the RHS of this rule is something new, `(inconsistent)`. It is equivalent to `(= 1 2)`. It is simply an explicit contradiction. Here is the new rule:

```
(occur (pickup *rob *obj) *t)
(state (on *thing *obj) *t)
                          ->    (inconsistent)
```

Here we have added a rule specifying a constraint between an action and a state, unlike earlier rules that constrain only states. Action constraints and state constraints are similar to the capability and legality constraint concepts that have been discussed elsewhere [Joslin and Roach, 1986].

It is interesting that in this model, inaction is represented vacuously. In other words, there is no need for a `wait` predicate, or its equivalent as is employed by many planning systems, since inaction is implicit in the interval between other actions.

Another point that should be made is that while this system is very basic, it is this very simplicity that makes it suitable for building representation methods for more complex concepts in a building block fashion, while keeping a conceptually simple underlying structure. For instance, we may want a system in which we can talk about events that take time. These are intervals in the underlying theory but it may be useful to speak of events in these terms for purposes of stating goals or reasoning about time or resources. In such a case, it should be easy enough to build up axioms about these meta-events or durative events in terms of the underlying instantaneous events and temporal intervals. In fact, Galton's Aspect Logic [Galton, 1984] is just such a system.

Our system is a simple basis upon which much more complex structures relating time, events, and states can be constructed. We believe that it is generally useful because we have chosen an appropriate set of assumptions on which to base our model for time. Our assumptions are both intuitive and make sense from a planning perspective.

There has been recent work in temporal logic [Van Benthem, 1983 and Turner, 1984] toward a unification of the concepts of time and events. They take events, and relationships between them, as primary, and then use the resulting structure to build up definitions of time instants and intervals. Our instantaneous events are simpler logically than their interval events. That difference notwithstanding, our system fits their description for a time/event model. To see the equivalence, we have to change viewpoint somewhat. One key fact to emphasize here is that we are not concerned with time points that lie between events. They are never even named by the system. This shows how closely the time points are tied to events. Our system does not define either as primary.

Turner axiomatizes his version of an event based temporal model in [Turner, 1984]. Using his work as an example to follow, we can go on to show that his definitions 4.1, 4.2, and his theorem 4.3 are valid in our system as well. Since we have not defined the temporal overlap operator, O, he defines for events, we will now do so. Because of the instantaneous nature of our events, the relation reduces to simple temporal identity, as follows. We will be using his temporal precedence operator <, which is defined between events instead of time points. If we use < between two events, read it as the operator defined below. In the following formulas, e_{t_i} represents an event at time t_i.

(Def O) $e_{t_1} \mathrm{O}\, e_{t_2} \equiv t_1 = t_2$

(Def <) $e_{t_1} < e_{t_2} \equiv t_1 < t_2$

Next we want to show, although not rigorously, that our system satisfies postulates (E1) - (E7) from his definition 4.1. Here are his postulates (E1) - (E7).

(E1) $e_{t_1} < e_{t_2} \supset \sim(e_{t_1} \mathrm{O}\, e_{t_2})$

(E2) $e_{t_1} < e_{t_2} \wedge e_{t_2} < e_{t_3} \supset e_{t_1} < e_{t_3}$

(E3) $e_{t_1} \mathrm{O}\, e_{t_2} \supset e_{t_2} \mathrm{O}\, e_{t_1}$

(E4) $e_{t_1} \mathrm{O}\, e_{t_1}$

(E5) $e_{t_1} < e_{t_2} \supset \sim(e_{t_2} \mathrm{O}\, e_{t_1})$

(E6) $e_{t_1} < e_{t_2} \wedge e_{t_2} \mathrm{O}\, e_{t_3} \wedge e_{t_3} < e_{t_4} \supset e_{t_1} < e_{t_4}$

(E7) $e_{t_1} < e_{t_2} \vee e_{t_1} \mathrm{O}\, e_{t_2} \vee e_{t_2} < e_{t_1}$

Using the definitions for temporal overlap and precedence given, the proof of each of these is trivial. For example, (E1) can be shown as follows. From $e_{t_1} < e_{t_2}$ we can conclude that $t_1 < t_2$ from (Def <). Since our time is a linear total order, $t_1 \neq t_2$, and therefore $(e_{t_1} \mathrm{O}\, e_{t_2})$ by (Def O).

His definition 4.2 defines temporal instants in terms of events as maximal subsets of pairwise overlapping events. Since our events are instantaneous, we too can define temporal instants in this way. Each instant is the overlap of the event at that instant with itself (or any other event that happens at the same time). His theorem 4.3 then states that the instants defined in this way are ordered temporally in a strictly linear fashion. A proof of this in our system is also trivial, since there is a one-to-one mapping from events to instants derived in our system, and the temporal precedence relation is defined to be the same. This exercise has shown how closely our system fits the framework these researchers are developing for event-based temporal logics and event structures. The ease with which we showed the match is partially a result of the common assumptions about time and events, and partially from the instantaneous events we use. We believe, however, that the equivalence would still be valid if our system were extended to directly represent events with a duration.

4.5. Causality

The last issue to discuss in this chapter is causality. There is far too much literature on causality to survey in this thesis. We have decided on a commonsense definition for causality in a problem solving system. To our system, a cause is simply a sufficient condition[1]. This makes sense in terms of planning systems because sufficient conditions are what a planning agent must create in order to make a goal true. We also assume that every change in the world has a cause. Here are these two last assumptions.

- A *cause* for a predicate is a sufficient condition for that predicate to become true, or to become false in the case of a negated predicate
- Every change (in the truth value of a predicate) in the world has a cause

A sufficient condition is written as a predicate logic expression, so any predicates in the system can be used to describe a sufficient condition. These include facts about time, states and events (changes). Based on these assumptions, there are a few general rules about the behavior of the world that our system follows. First, it deduces from a `state` whose negation it can derive using persistence that the `start` of that `state` occurred

[1] We are implicitly assuming a deterministic world. It is clear that a coin flip is not a sufficient condition for a result of heads, although exactly what is a sufficient condition is difficult to say. Our system will have to be extended if it is to handle nondeterministic events such as this example. The believability rating provided by the modal category mechanism of WIF may be useful in this regard.

since the stop that started its negation.[1] Next, it deduces that there must be an event to cause every start or stop. This is sensible unless we think that random changes can occur in the world without causes. For the start and stop of each predicate, we require an explicit statement of each sufficient condition.[2] While it is true that these must be enumerated, only *direct* causes need be listed. It is not actually necessary to specify the pair of causal rules for every predicate. Any predicate from which a different predicate that does have a causal rule can be derived need not have its own causal rules. Below are the corresponding WIF consistency rules. The last one is the general form for the causal rule for a predicate. In an implementation, there are two for each predicate: one for the start and one for the stop.

```
(start (pred_name *args) *t)
(<> *t t0)
(not cause_of1,pred-name-start *t)
(not cause_of2,pred-name-start *t)
...
(not cause_of3,pred-name-start *t)          ->     (inconsistent)

(stop *fact *t)
(<> *t t0)
(not cause_of1,pred-name-stop*t)
(not cause_of2,pred-name-stop*t)
...
(not cause_of3,pred-name-stop*t)            ->     (inconsistent)
```

A rule for a start of a particular predicate, pred_name, for instance, states that if we know that the predicate becomes true (at some time other than the initial state), and

[1]There is a corresponding dual for this rule that derives that a (not (state *fact *t)) began with a (stop *fact *t1) since the most recent (start *fact *tprevious). Notice that the persistence rule given earlier is very closely related to these two rules. They deduce the existence of a cause from an apparent contradiction between a known fact, and persisting facts. Persistence derives the truth value of a fact based on *lack* of any event affecting the fact.

[2] Remember that this is a planning system. It is interested in creating a future possible world in which its goals are true, presumably as a (possibly indirect) result of its own action. It is therefore concerned only with causes that it understands (those that are enumerated). Worlds in which things change for no apparent reason are not useful for a planner, since it cannot act to create those worlds.

that none of the known possible causes, $\texttt{cause\_of}_{\texttt{i,pred-name-start}}$, for the `start` of `pred_name` are true at that time, then the world is inconsistent. Each of the `cause_of` conditions may be an arbitrarily complex expression if desired. If this rule successfully instantiates, WIF then restores consistency by creating three possible worlds (in the above example rules) : each containing the negation of a LHS clause. Since the <> cannot be negated because of its low modal category, and `start` is a hypothesis, the three worlds contain $\texttt{cause\_of}_{\texttt{1,pred-name-start}}$, $\texttt{cause\_of}_{\texttt{2,pred-name-start}}$, and $\texttt{cause\_of}_{\texttt{3,pred-name-start}}$. This gives us three possible worlds, each containing a sufficient cause for the `start` of `pred_name`. Notice that `cause_of` is a relation, in the mathematical sense, between facts.

Since the causes for each predicate are enumerated, for a finite number of predicates the causal rule schemas given above are of the first order logical form:

$$\forall j\text{: [start\_pred}_j\text{(args,t)} \wedge \forall i\text{:\~{}cause\_start\_pred}_{j,i}\text{(args, t)} \supset \text{false]}$$

or, equivalently,

$$\forall j\text{: [start\_pred}_j\text{(args,t)} \supset \exists i\text{:cause\_start\_pred}_{j,i}\text{(args, t)}$$

Each event in a domain axiomatization can be represented in the following form:

$$\forall e\text{: [occur\_event}_e\text{(args,t)} \supset \xi_e\text{(args, t)}$$

where ξ_e(args, t) is a conjunctive expression containing the direct preconditions and results of the event. It needs no disjunctions because to use them would be to express a set of precondition/result mappings. Because we need only the direct preconditions and results, an event can be defined with a single description. The entire rule can be written equivalently as a set of implications, all with the same antecedent as above, and one of the conjuncts on the right. If we rewrite the rules in this first order form, they are in the separable form required for circumscription, so we can perform a delimited circumscription[1] of the state predicates found on the RHS of these rules. The result is equivalent to our causal rules. This is because we have, as stated before, assumed that we know about all the possible causes for a change.

The following example of the two causality rules for the on predicate from the Blocks World illustrates how the causality rules are written.

```
(start (on *obj *loc) *t)
(not (occur (putdown *obj) *t))
                              -> (inconsistent)

(stop (on *obj *loc) *t)
(not (occur (pickup *obj) *t))
                              -> (inconsistent)
```

In each case, there is only one possible cause. That is a result of the simplicity of the Blocks World. When `(start (on a b) t1)` is hypothesized in WIF, the first rule is found and instantiated. The result is `(occur (putdown a) t1)`, unless it was true already, in which case the LHS does not fully instantiate and the rule does not apply. In a more complex world, there could be several causal conditions negated on the LHS, and the conditions could be more complex expressions.

An interesting point to raise here is the issue of intentionality. For a planning agent to understand the world, he must have information about causality that is more than just the actions he can perform. For example, let us say that an agent needs light to do something. While he may be capable of turning on a light, he should also know that being outdoors in the day will solve the problem of getting light. The agent knows that daytime can 'cause' light even though he cannot control it. The difference between actions the agent can

[1] The expression for the delimited circumscription of a set of separable predicates is given by Theorem 6.15 of [Genesereth and Nilsson].

perform and actions the agent cannot perform can be modeled by modal categories, or by a more detailed model of the agent's capabilities.

4.6. Domain Modeling Guidelines

Following are some conventions and techniques we use for modeling domains. They are not part of the theory but we have found them to be useful for representing similar information found in many domains. This allows models for particular domains to be constructed more quickly, and it allows some of the consistency rules in the system to be domain-independent. These conventions and techniques are concerned with both the representation of a state of the world and of changes between states.

Objects in the domain of the application being implemented are represented by a hierarchical classification scheme. The classes give rise to the properties of the object. An object is an individual by virtue of being a class with only one member. The scheme is very flexible because it does not directly restrict membership in classes based on what classes an object belongs to already. Since class membership gives rise to properties, mutual exclusion occurs among properties and values of properties, instead of among classes. Subclasses inherit the properties of all of their parent classes, of which there may be any number.

To show the advantages of this paradigm, take the case of the classes 'daughter' and 'father-in-law'. These two classes are obviously mutually exclusive, but to include a rule specifically stating so implies that there are a lot of other rules stating the inconsistency of being in various combinations of relationships. The number of interaction rules increases with the square of the number of relationships represented. In this system, one simply states:

```
    (attribute *person sex *val1)
```

```
(attribute *person sex *val2)
                              -> (= *vall *val2)

(isa daughter *a)
                              -> (attribute *a sex female)

(isa father-in-law *a)
                              -> (attribute *a sex male)
```

The first rule ensures that only one value of the attribute sex is allowable for any particular individual. The next two allow the derivation of the sex attribute from the relationship of the person. If, in one possible world, one person was both a daughter and a father-in-law, the first rule would make that world inconsistent. Extensions to this rule set are linear with the number of new relationships; one rule is added for each relationship for which the sex attribute is relevant. Therefore, with 1+n (where n is the number of relationships) rules, we can represent n^2 of the rules about combinations of relationships.

5. Planning

In this chapter we discuss planning in general, and how our system creates plans in particular. By planning we mean the process of finding a sequence of actions that, when executed over some time interval, should, to the best of the planner's knowledge, produce a desired goal state. Therefore, in order to plan we need to know about the current state of the world, and the desired future state. From this basis we can extrapolate, performing actions to cause the desired changes. We also need knowledge about how the world operates in general. These types of information and their representations were discussed in the previous chapter. Given this information, a planning system produces a plan. The result of planning is, minimally, the sequence of actions leading to the goal. We will call this ordered sequence of actions the plan, although there is a great deal of other information that can be produced during planning that is useful during execution. Our system produces this information for later use by the planning agent. We produce a nonempty (if a plan exists) set of causally connected state sequences, with the first state in the sequence(s) corresponding to the initial state, and where the goal is true in the last state of each sequence.

We will start with a chronological discussion of the most important previously developed systems. During this discussion, some famous planning problems that have spurred the evolution of problem solvers will be introduced in the appropriate places. Then we introduce a new type of planning problem, the indirect goal problem, which previous systems cannot solve. We discuss our solution to it and why we can accomplish this where other systems cannot. The chapter finishes with an explanation of our planning procedure, and a couple of examples that illustrate its operation.

5.1. Previous Work in Planning

Planning, or problem-solving, is one of the primary research areas in AI and has received much attention. Like most of the rest of AI, it is still unsatisfactorily solved despite the volume of research already conducted, although progress has been made. This section reviews, in a primarily chronological order, the major previous work relevant to this thesis.

GPS, or General Problem Solver, was an early theorem proving system that could be applied to planning [Ernst, 1969]. It introduced the technique of goal ordering to handle multiple goals. The idea behind goal ordering is to solve each goal independently, and then find an order in which the resulting plans can be concatenated together to get a plan to solve the composite goal.

One of the first successful systems designed for planning was STRIPS [Fikes and Nilsson, 1971]. It modeled one agent in a simple world with boxes and rooms. Actions were represented by precondition and add-delete lists. If all of the facts in the precondition list were true, then the action could be legally performed, and then the facts in the add-delete list were added to/deleted from the world to model the changes resulting from the action.

Building on GPS and STRIPS, efficiency gains were made by LAWALY [Siklossy and Dreussi,1973], the first hierarchical planner. It represented actions on a hierarchical scale of increasing abstraction. It planned with very high-level operators first, and then created a more detailed plan for each subgoal. This improved on the efficiency of STRIPS and allowed much larger problems to be solved.

Further improvements were introduced by the NOAH planner [Sacerdoti, 1977]. It operated by delaying decisions about the ordering of actions as long as possible. NOAH was developed to solve a class of problems Sacerdoti called nonlinear problems [Sacerdoti, 1975], because they could not be solved in linear time, or even at all, by goal ordering. NOAH was capable of solving them. We fully discuss nonlinear problems in the next section. We will just say here that they are characterized by goals that cannot be solved independently due to interactions between the actions performed in the plans for each individual goal.

NONLIN [Tate, 1977], an extension of NOAH, added backtracking to NOAH's capabilities. It could also solve nonlinear problems while still solving linear problems in linear time. This increased the capability of NOAH by allowing it to solve problems that required backtracking.

An important problem faced by realistic planning systems is the representation and use of incomplete and/or incorrect knowledge. The systems we have discussed already all assumed that the planning agent had complete and correct knowledge of the world. There are a number of approaches to representing uncertain knowledge in the literature, one of which is the uncertainty approach employed by Shortliffe, Davis, and Buchanan's MYCIN bacterial infection expert system. Another technique for handling uncertainty is to store justifications (or endorsements) for facts [Cohen, 1985], so that the knowledge base is actually a large dependency graph. This allows changes to propagate, and allows the strength of a belief to be derived from the strength of supporting beliefs. Consistency maintenance is the other major technique for handling incorrect knowledge [Roach, Eichelman, and Whitehead, 1985]. We have extended this approach to represent incomplete factual knowledge with a possible worlds approach. As far as actual planning systems that can handle incomplete knowledge, only the Know-No-Know system [Roach, 1975] had actually implemented this capability until the system we describe in this thesis was developed. Know-No-Know could create plans, and then repair them later when they were found to be incorrect because of an incorrect world model.

David Chapman created the TWEAK planner [Chapman, 1985] as an enhancement of the previous methods in the field. His technique was special however, because he defined it so that he could prove that it was correct and complete. He could also analyze the previous systems in terms of his system's definitions.

Another technical result is found in [Joslin and Roach, 1986]. They proved that it was impossible to create a planning system that can solve arbitrary planning problems in linear time. They also include a careful analysis of goal ordering and some extensions to goal ordering that have been proposed.

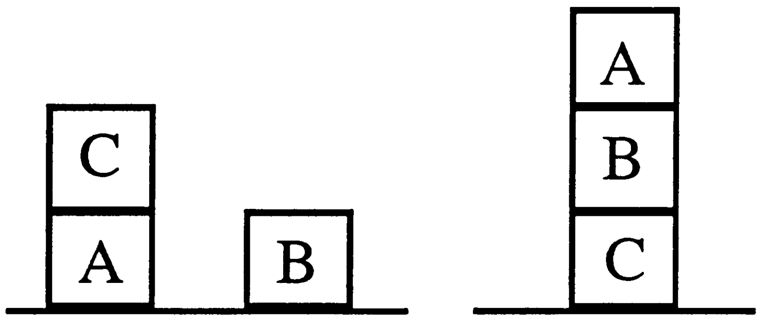
In [Roach, 1987] and [Joslin and Roach, 1986] the idea of applying hypothetical reasoning to planning was proposed, along with some preliminary ideas. This thesis has grown most directly from these papers, in that they provide the planning focus for this thesis. We have also drawn from several other areas in this research.

There has also been a great deal of work on the frame problem as it relates to planning systems. This is dealt with in Chapter 7. Many of the systems introduced here are also discussed in Chapter 7 for comparison to each other and to our system.

5.2. Goal Ordering and Nonlinear Problems

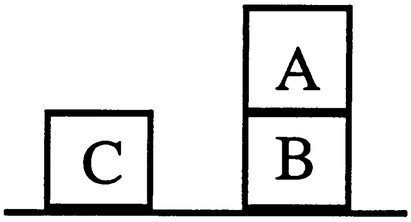
The goal ordering approach, which may or may not be hierarchical (breaking goals down to subgoals for individual solution), makes a major assumption that is not always valid. This assumption is that a set of goals can be solved by solving each goal independently, and then concatenating the plans for each goal into one complete plan. This approach will not work because it cannot handle interactions between the plans created for each goal. There may be no way to order the individual plans so that their concatenation produces a valid plan for the entire set of goals.

Problems with this property are known as nonlinear problems. For example, consider the classic CAB blocks world problem. There is only one type of action possible, the `move` action, which takes a block from where it is to another block or to the table. A block may not be moved if it has a block on top of it, and a block may not be moved onto a block that already has a block on top of it. The initial state is illustrated below on the left. The goal state, `(on a b)`, `(on b c)`, is shown below on the right.

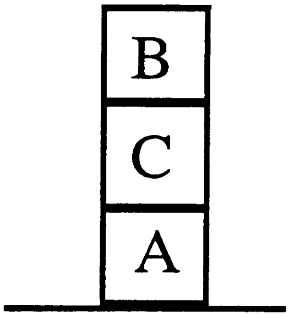


Initial State Goal State

In attempting to solve this by goal ordering, we first create plans for each goal. Both are very simple. The plan for `(on a b)` is `(move c table)`, `(move a b)`. The plan for `(on b c)` is `(move b c)`. Now we have to execute one plan and then the other to get to the goal state. But if we execute `(move c table)`, `(move a b)` first, we cannot execute `(move b c)` and vice versa. These situations are illustrated below.



Direct Solution of
`(on a b)`



Direct Solution of
`(on b c)`

From this example, it is clear that goal ordering is not sufficient for a general purpose planning system. Sacerdoti's NOAH planner [Sacerdoti, 1975] was the first published system to successfully solve this type of problem in an efficient manner.

5.3. The Indirect Goal Problem

We have identified a class of problems that we call indirect goal problems, that cannot be solved by previous planning systems, even those capable of handling nonlinear problems. This class of problems is characterized by goal statements that are not direct results of any action by the agent. These problems are not even possible to specify for some types of planners, however, because some planners require that all consequences of every action be specified directly. Therefore, there are no indirect consequences possible, and so if a predicate can be made true at all, there must be an action that can make it true directly. In some of the planners that are being developed today, ours for instance, and some past problem solvers, such as GPS, there are indirect effects of actions, and these indirect effects are the core of the problem. A system such as GPS looks for differences between the initial and goal state, and then tries to find actions to fix the differences. If no action directly affects the predicate for which there is a difference, GPS cannot find an action to apply to eliminate the difference.

A system such as GPS can solve a problem of this type by redefining the domain so that the predicate of interest is a direct result of an action. In order to create a GPS type planner that could solve any problem in the domain, however, the domain would have to be defined so that all predicates could be made true as the direct result of an action. Furthermore, all possible useful applications of an action would have to be modeled, reducing GPS to a STRIPS-type representation of the domain, where a separate rule is necessary for every case in which action is applicable. This is obviously not acceptable. Sacerdoti's NOAH planner [Sacerdoti, 1977] also required an operator for each goal predicate, while some other planners handle these problems as special cases with extra procedural code. No other planners can solve these problems within their own planning paradigm.

Our system can solve this type of problem because we do not look only for actions that make our intended goal true. There are also the other domain rules describing the relationships between predicates. Using these, we can derive other information about the goal state. As long as we have fully described the predicate in terms of other predicates, WIF will ensure that the goal state *as a whole* is consistent with our causal rules. We are really enforcing causality for an entire world state, not just for individual predicates.

For a very simple example of this type of problem, consider the blocks world predicate `over`, which is the transitive closure of the `on` predicate. Typical domain rules state that `on` is the result of the `move` action. An object is `over` another object if it is `on` the object or if it is `on` something that is `over` the object. Using this fact, we can deduce from `(over a b)` goal that either `(on a b)`, or `(on a *other)` and `(over *other b))`. The second case will create a number of possible worlds, limited by the legal stacks of objects in the domain. Anyway, both cases resolve to on predicates that can be planned for with move actions.

This type of problem[1] will become more important as the domains modeled become more complex, because more complex domains require a simpler rule scheme in order for the rules to be determined and written reliably and in reasonable space. It is important that future general planning systems are also capable of handling these problems, as otherwise the problems that they are capable of solving will be limited. The more complex the systems become, the more dependence among predicates will be necessary, and the greater the limitation will be.

5.4. The Planning Algorithm

All planning (except persistence and the mechanics of creating new time points, as we have discussed already) is performed by WIF. The rules we have provided enable it to 'plan' simply by generating the consistent possible worlds. We supply WIF with an initial world state and the consistency rules, including all of the event and causality rules. Then, when given a goal as a hypothesis, WIF uses these rules and facts to create the set of consistent possible worlds that include the goal. Each of these worlds is a set of facts about a possible future timeline. Since the consistency rules include rules that ensure every fact about a state has a cause, and that each cause is an event, there is a causal chain of events leading to the goal state included in the facts for each possible world. Presumably the events are actions performed by the agent, although this is not necessary. If there were multiple agents and another agent can be drafted into doing parts of the plan, then the planning agent may not be performing all of the actions in the plan. The course of time

[1] Notice that indirect goal problems and nonlinear problems are not mutually exclusive classes.

could also be regarded as an agent, doing such actions as 'night falls' [thud], if darkness were necessary for some action by the agent. This system is interesting precisely because there is no explicit algorithm; there are only general rules about the world. We have not created a specialized algorithm just for planning, we have provided rules and facts to an existing consistency maintenance algorithm and interpreted the result as a plan (actually a set of plans, since each possible world from WIF contains a plan).

A useful by-product of the 'planning' performed by WIF is the set of facts derived during processing. Each of these facts is an expectation about the world if all goes according to plan. Any explicit preconditions for an action will be part of the plan. During execution of the plan, testing perceptions against this extra information with WIF will immediately alert the agent to problems with his plan. For an example of this process in action, refer to section 7.3.1 concerning the qualification problem.

We have had to include a limit on the number of actions an agent can perform. Since we have allowed dense time, WIF could try to produce a world with an infinite number of actions by the agent. This is only a problem because we have no resource reasoning methods implemented to handle temporal restrictions between states and events. This restriction becomes unnecessary as soon as we include rules that restrict how many things an agent can do at the same time, and specify how long various actions take to perform.[1]

In effect, the limit on the number of actions that an agent can perform is a very crude approximation of the fact that actions take time and so only a finite number can be performed in a finite time period. The more realistic rules for resource reasoning that

[1] Whether there is an arbitrary numerical depth limit or a resource reasoning mechanism limiting the search space, the plans resulting from the process will always have a finite number of steps and could therefore be mapped to a integer model. During planning however, the system has access to a dense time model.

implicitly create a limit to agent actions, are naturally expressible as constraints, which are very straightforward to state in WIF. Our conclusion is that all models of the real world in our system are composed of non-zero-length intervals. Our representation of states of change makes this possible. This point will become important again later, when we discuss completeness.

Our 'algorithm' is very different from others that exist in the published literature on planning. We do not start at either the initial or goal state and work toward the other. Our system works from the inside out, filling in facts that it can deduce from the rules and other facts. The facts are not derived in any particular order, and since all possible worlds are explored, the order is not important. Facts about several states can be derived before any of the states are completely described. There is no particular order in which facts or states are considered. Of course, practically speaking, WIF and the rule and fact sets implicitly defines an order on the deduction process, but the order does not change the result (although it may change the time needed to get the result). The next two sections clarify our planning process with annotated execution traces of two examples. The domain rules in the examples have been condensed from the rules in Appendices A and B so that the examples are easier to follow.

5.4.1. The Blocks World CAB Problem Solution

In this section, we will trace the operation of WIF producing a plan for the Blocks World nonlinear CAB problem. We start with the rules used in the trace, numbered for reference during the trace. At various points during the process, new timepoints are created. We will write a timeline (the fully ordered timepoint set) at the points during the trace within braces as follows: `{t0,...,t1}`. The initial state time is `t0` and the goal state time is `t1`. The initial and goal state predicates are then listed, followed by the trace itself.

The `earlier` predicate is our explicit < relation, created to allow us to insert points at any point in the timeline when necessary. Another predicate introduced here is the `during` predicate. It is also implemented by a PROLOG rule. When invoked with `(during *start *new *end *before *after)` with `*start` and `*end` as bound variables, it creates a new timepoint `*new`, and returns all of the possible minimal subintervals of `(*start, *end]` in `*before` and `*after`. Each minimal interval creates a new possible world with a different total order on the timepoints. The `persist` predicate used in the rules is an explicit call to the PROLOG persistence rule given for `states`. It is used because it returns a modal category of 0, and therefore cannot be negated, and so that the time of the event providing the persistence information can be obtained.

WIF's database starts with the initial and goal state facts as given below, before the trace. The trace consists of successful rule instantiations. The facts that are added to the database as a result of the inference rule instantiations are marked with a •. Facts whose negations are added to the database are marked with a ••. A great many rules apply to most new hypotheses, but only those that do something significant have been included here. Others, such as transitivity for `over` have been left out for brevity.

Domain Rules

```
(state (on *a *b) *t)                                        (1a)
(<> t0 *t))
(persist (not (state (on *a *b) *t)) *tprev)
     ->    (get_during *tprev *tnew *t *before *after)
           (earlier *before *tnew)
           (earlier *tnew *after)
           (start (on *a *b) *tnew)

(not (state (on *a *b) *t))                                  (1b)
(<> t0 *t))
(persist (state (on *a *b) *t) *tprev)
     ->    (get_during *tprev *tnew *t *before *after)
```

```
                  (earlier *before *tnew)
                  (earlier *tnew *after)
                  (stop (on *a *b) *tnew)

    (start (on *a *b) *t)                                        (2a)
    (not (occur (move *a *b) *t))
         ->   (inconsistent)

    (stop (on *a *b) *t)                                         (2b)
    (not (occur (move *a *place) *t))
         ->   (inconsistent)

    (occur (move *a *b) *t)                                      (3a)
    (state (on *any *a) *t)
         ->   (inconsistent)

    (occur (move *a *b) *t)                                      (3b)
    (state (on *any *b) *t)
    (not (isa table *b))
         ->   (inconsistent)

    (state (on *a *b) *t)                                        (4)
         ->   (state (over *a *b) *t)

    (state (over *a *b) *t)(state (over *b a) *t)               (5)
         ->   (inconsistent)
```

Initial State {t0}

```
    (state (on c a) t0)
    (state (on a table) t0)
    (state (on b table) t0)
```

Goal State {t0,t1}

```
    (state (on a b) t1)
    (state (on b c) t1)
```

Execution Trace

```
    (state (on a b) t1)                           (rule 1a)
    (persist (not (state (on a b) t0)) t1)
    (earlier t0 t1)
         ->   (during t0 ta t1 t0 t1)•
              (earlier t0 ta)•
              (earlier ta t1)•
              (start (on a b) ta)•                {t0,ta,t1}
```

From the change in states, we have deduced the existence of a time-point during the interval from `t0` to `t1` where the state `(on a b)` became true. We have created and inserted the new point into the temporal ordering and added the `start` of the fact to our knowledge.

```
(start (on a b) ta)                              (rule 2a)
(not (occur (move a b) ta))••
     ->   (inconsistent)
```

We have found a `start` without a known cause. Since only one cause is possible for the start of on, we add in and stay in one possible world. In general, there could be several negated causes on the LHS of the causal rule for `start` or `stop` of a predicate, which would result in several possible worlds, each containing one of the possible causes.

```
(occur (move a b) ta)                            (rule 3a)
(state (on c a) ta)••
     ->   (inconsistent)
```

We find that we cannot move block a unless c is not on a.

```
(not (state (on c a) ta))                        (rule 1b)
(persist (state (on c a) t0) ta)
(earlier t0 ta)
     ->   (during t0 tb ta t0 ta)•
          (earlier t0 tb)•
          (earlier tb ta)•
          (stop (on c a) tb)•                    {t0,tb,ta,t1}
```

We deduce and create the necessary `stop` and its timepoint.

```
(stop (on c a) tb)                               (rule 2b)
(not (occur (move c *place) tb))••
     ->   (inconsistent)
```

The negation of the above condition implies the occurrence of `(move c a)` or `(move c b)` or `(move c c)` or `(move c table)` at `tb`. Each becomes a separate possible world. But the first is impossible by rules 4 and 5. The third is also impossible by rules 4 and 5. The second is impossible by rule 3b, in conjunction with `(on a b)`. The fourth is the only world that does not get ruled out. We will now continue with it.

```
(occur (move c table) tb)•
```

Considering the second goal,

```
(state (on b c) t1)                              (rule 1a)
(persist (not (state (on b c) t0)) t1)
     ->   (during t0 tc t1 *before *after)•
          (earlier *before tc)•
          (earlier tc *after)•
          (start (on b c) tc)•
```

In this case, `during` can put `tc` in three places on the timeline, resulting in the three timelines `{t0,tc,tb,ta,t1}`, `{t0,tb,tc,ta,t1}`, and `{t0,tb,ta,tc,t1}`. In the first case, after a several rules are applied, it finds that `b` could not have been put on `c` before it was moved off `a`. In the third case, it finds that `b` cannot be moved at all, because `a` is on top of it. Therefore, the only surviving possible world from the above rule is:

```
     ->    (during t0 tc t1 tb ta)•
           (earlier tb tc)•
           (earlier tc ta)•
           (start (on b c) tc)•                {t0,tb,tc,ta,t1}

(start (on b c) tc)                            (rule 2a)
(not (occur (move b c) tc))••
     ->    (inconsistent)
```

No further rule applications add anything of interest to the result, or rule out this possible world. Therefore, the final answer for this plan is the possible world with the facts listed below and the timeline `{t0,tb,tc,ta,t1}` described by `earlier` predicates. Other facts have been derived about the world but were never used so are not included here.

```
(state (on c a) t0)
(state (on a table) t0)
(state (on b table) t0)

(stop (on c a) tb)
(occur (move c table) tb)

(start (on b c) tc)
(occur (move b c) tc)

(start (on a b) ta)
(occur (move a b) ta)

(state (on a b) t1)
(state (on b c) t1)
```

5.4.2. An Indirect Goal Problem in Blocks World

In this section we present our system's solution to an indirect goal problem. The problem is a simple problem involving three blocks in Blocks World. The problem is described below by the initial and goal states. The domain rules from the previous section also apply here. Additional rules are listed below. There is a limit of 2 actions on the length of the plan, so worlds that contain longer plans are thrown out. A limit of 3 or higher also works, but results in more plans, containing less direct solutions.

Domain Rules

```
(state (over *a *b) *t)                              (6)
(not (state (on *a *b) *t))
     ->   (isa thing *place)
          (state (on *a *place) t1)
          (state (over *place *b) t1)
```

Initial State

{t0}

```
(state (on a b) t0)
(state (on b table) t0)
(state (on c table) t0)
```

Goal State

{t0,t1}

```
(state (over c b) t1)
```

Execution Trace

```
(state (over c b) t1)                                (6)
(not (state (on c b) t1))
     ->   (isa thing *place)
          (state (on c *place) t1)
          (state (over *place b) t1)
```

Since c is over b, it must either be on b, or on something else that is over b. The planner generates two possible worlds here: c on b (from the LHS) and c on a on b (from the RHS). In the first case, the planner finds that c moves to b, which results in both c and a being on top of b, so it tries to move a off of b before c is moved to a, but this results in too many events, so that world is thrown out. The second case is the correct path. That instantiation is:

```
(state (over c b) t1)                                (6)
(not (state (on c b) t1))
     ->   (isa thing a)
          (state (on c a) t1)•
          (state (over a b) t1)
```

Continuing:

```
(state (on c a) t1)                                  (rule 1a)
(persist (not (state (on c a) t1)) t0)
(earlier t0 t1)
```

```
        ->    (during t0 ta t1 t0 t1)•
              (earlier t0 ta)•
              (earlier ta t1)•
              (start (on c a) ta)•              {t0,ta,t1}
```

From the change in states, we have deduced the existence of a time-point during the interval from `t0` to `t1` where the state `(on c a)` became true. We have created and inserted the new point into the temporal ordering and added the `start` of the fact to our knowledge.

```
   (start (on c a) ta)                          (rule 2a)
   (not (occur (move c a) ta))••
        ->    (inconsistent)
```

We have found a `start` without a known cause, so we add a possible cause (the only possible cause for on in blocks world) to the fact base. No further rule applications add anything of interest to the result, or rule out this possible world. Therefore, the final answer for this plan is the possible world with the facts listed below and the timeline `{t0,ta,t1}` described by `earlier` predicates. Other facts have been derived about the world but were never used so are not included here.

```
   (state (on a b) t0)
   (state (on b table) t0)
   (state (on c table) t0)

   (start (over c b) ta)
   (start (on c a) ta)
   (occur (move c a) ta)

   (state (over c b) t1)
```

6. Theoretical Results

The theoretical framework for the world model on which our system is based consists of the extension $\mathbf{K}_{ld}$ of Prior's Minimal Tense Logic $\mathbf{K}_t$, a problem solving algorithm based on consistency maintenance, and a mixture of domain independent and domain dependent consistency rules representing causality. This system is sufficient for 'understanding' the physical world to the extent that meaningful problems can be solved within the representation by the mechanisms given. Extending this system to handle a construct similar to Galton's event-radicals [Galton, 1984] might make the system much more intuitive for representing events but we chose the simpler formalism for its relative simplicity and existing body of theorems. Such an extension would not increase the expressive power of the system according to [Galton, 1984] but it could make the modeling of multiple introspective agents more straightforward by allowing the creation of models that deal directly with events that take time. An additional benefit of the basis in consistency maintenance is that we can deal more directly with some of the other issues such as perception and unexpected information and its scope.

In this chapter, we present a formal logical basis for this work, and explain the semantics of the logical model to show its generality for handling applications involving problem solving systems. The tense logic $\mathbf{K}_{ld}$ is introduced in order to describe the model theory corresponding to the output of our planning system in the familiar terms of the modal system S_5.

6.1. The Axiomatization - Tense Logic

This section gives a complete axiomatization of the tense logic K_{ld}, the system defined by the assumptions about time discussed in Chapter 4. They are used in 6.2 to derive the modal system that provides a model theory for our system.

This tense logic axiom system is an extension of the minimal event logic known as K_t: standard propositional logic, extended by the addition of four axioms, 2 definitions, two rules of inference, and two tense operators, P and F. Pp means that the proposition p has been true, Fp means that p will be true. The symbol p stands for any proposition about a state. The symbol α stands for any axiomatically derivable theorem. The inference rules, definitions, and axioms for K_t are:

(Prop)	The axioms and rules of inference of Propositional Logic
(Def H)	$H\alpha \equiv \sim P \sim \alpha$
(Def G)	$G\alpha \equiv \sim F \sim \alpha$
(Inf H)	$\alpha \supset H\alpha$
(Inf G)	$\alpha \supset G\alpha$
(A1)	$G(p \supset q) \supset (Gp \supset Gq)$
(A2)	$H(p \supset q) \supset (Hp \supset Hq)$
(A3)	$p \supset HFp$ (or $FHp \supset p$)
(A4)	$p \supset GPp$ (or $PGp \supset p$)

These axioms form the system K_t [Galton, 1984], about which many results have already been proven [McArthur, 1976], including consistency and completeness [Galton, 1984]. Gp means that p is now and will always be true, and Hp means that p is now and always was true. (Inf H) and (Inf G) allow us to state that any theorem from propositional calculus is a theorem at all times. (Def H) and (Def G) define H and G as the duals of P and F respectively. (A1) and (A2) state that if an inference is always valid, then the consequence is always true if the antecedent if always true. (A3) means that if p is true, then it has always been the case that p will be true. (A4) states (A3)'s dual. A dual of a sentence in tense logic is the same sentence with all P's replaced with F's and vice versa, and all G's replaced with H's and vice versa. Any theorem's dual is also a theorem since the axioms are duals of each other.

Given these definitions, the system must be extended to formalize the assumptions that we made in Chapter 4 about the nature of time. The following axioms make those assumptions explicit:

(A5)	$Pp \supset PPp$	(dense time)
(A6)	$Fp \supset FFp$	(dense time)
(A7)	$PPp \supset Pp$	(transitive time)
(A8)	$FFp \supset Fp$	(transitive time)
(A9)	$Fp \wedge Fq \supset F(p \wedge Fq) \vee F(p \wedge q) \vee F(Fp \wedge q)$	(linear time)
(A10)	$Pp \wedge Pq \supset P(p \wedge Pq) \vee P(p \wedge q) \vee P(Pp \wedge q)$	(linear time)

The time structure is neither reflexive nor symmetric. (A5) and (A6) make the rational number nature of time explicit by specifying that a new time-point exists between any two others. (A7) and (A8) are the standard tense-logical formalisms for a transitive temporal precedence relation. (A9) and (A10) force a total order on time points. A state is defined as a maximal homogeneous interval, which is allowable since an instant is certainly homogeneous. Since events are state transitions, they are instantaneous as discussed already.

These axioms define $\mathbf{K}_{ld}$ [Rescher and Urquhart, 1971], a standard system with well defined properties.[1] The standard semantics for tense logics treat propositions at time-

[1] This set of axioms fails to specify the intended model of time completely. There is no distinction made here between the intended dense, antisymmetric, total order without eternal recurrence, and circular or cyclical world histories. This distinction cannot be made in a tense logic. An extension such as metric tense logic or a first order temporal logic is required. We chose to accept this ambiguity in return for the simplicity of the resulting axioms.

points, not over intervals as we do. This is permissible, as others [Humberstone,1979] have defined interval semantics for tense logics. The point is to provide a standard tense logical system embodying the original assumptions as a basis for further development.

6.2. The Model - Modal Logic

The tense logical axiom system laid out in 4.2 is an S_5 modal system, the system in which the accessibility relation between possible worlds is an equivalence relation: transitive, symmetric, and reflexive. This is obviously not the same relation as the temporal precedence relation <, which is transitive, antisymmetric and irreflexive. The accessibility relation is between entire timelines, while temporal precedence holds between instants (or intervals, following Humberstone's semantics [Humberstone, 1979]) in a timeline. All timelines passing through the same initial state are related (in the same equivalence class). An equivalence class is, therefore, the set of all possible future worlds of a given initial state. Multiple equivalence classes arise because of incomplete knowledge about the initial state. This gives us a class of accessible futures for each possible initial state, a very useful result for planning (or prediction for that matter). We conjecture that any general planning system must make at least the above assumptions about time, also resulting in an S_5 model. This is significant because S_5 is such a powerful system.

Showing that the axioms for $\mathbf{K}_{ld}$ of 6.1 do correspond to an S_5 model is straightforward. First, it is necessary to define the modal operators $\Box p$ (p is always true) and $\Diamond p$ (p is true sometime).

$$\Box p \equiv Hp \wedge p \wedge Gp$$
$$\Diamond p \equiv Pp \vee p \vee Fp$$

This is called the Megarian style of temporal necessity, for the Greek philosophical tradition espousing this view. For other views, see [Rescher and Urquhart, 1971].

Next is the construction of **M**, the smallest normal modal system. It consists of [Rescher & Urquhart, 1971]:

Rule of Inference:	if α then $\Box\alpha$	($R\Box$)
Axioms:	$\Box(p \supset q) \supset (\Box p \supset \Box q)$	(Ax 1)
	$\Box p \supset p$	(Ax 2)
Definition:	$\Diamond p \equiv {\sim}\Box{\sim}p$	($D\Diamond$)

(R) follows directly from (Inf H) and (Inf G). (Ax 1) follows directly from the definition of $\Box p$ and (A1) and (A2). (Ax 2) follows directly from the definition of $\Box p$. ($D\Diamond$) is easily shown true given the previous definitions of $\Box p$ and $\Diamond p$.

Extending this to S_5 is simply a matter of adding two axioms as follows [Rescher & Urquhart, 1971]:

$\Diamond\Box p \supset p$	(Ax 3)
$\Box p \supset \Box\Box p$	(Ax 4)

(Ax 3) follows directly from the given alternative forms of (A3) and (A4) and the definitions of $\Box p$ and $\Diamond p$. (Ax 4) is a consequence of (A5) and (A6) and the same definitions. The modal counterpart of (A7) and (A8), $\Box\Box p \supset \Box p$, follows directly from (Ax 2), so (A7) and (A8) do not add anything to the modal system. A rigorous version of the proof outlined above for the equivalence of $\mathbf{K}_{1d}$ and S_5 is given in [Rescher & Urquhart, 1971].

6.3. The Implementation Axioms

The implementation portion of this work proceeded in two stages. The first (Version A) was a preliminary attempt at a planner using consistency maintenance. The possible worlds produced by our version A system are equivalent, without proof here, to the possible worlds semantics of tense logic $\mathbf{K}_{li}$ (linear discrete time) and modal system **D**. See [Rescher and Urquhart,1971] for a detailed description of these systems. The second (Version B) is the system we have described throughout this thesis.

6.3.1. Version A Axioms

This axiom schema describes the initial version of the counterfactual planner. It describes a system in which time is modeled by a finite number of totally ordered points, of which each neighboring pair defines a left-open nondecomposable interval, over each of which a state proposition is either true or false. The representation for the truth of state proposition p over the interval $(t_i,t_{i+1}]$ is p_{t_i}. The proposition for an event at time t_i is e_{t_i}.

(1) $\forall t_i: i \in \{0,1,2,\ldots,n\}$
(2) $t_0 < t_1 < t_2 < \ldots < t_n$
(3) $\forall t_i, t_0 < t_i: p_{t_i} \equiv (p_{t_{i-1}} \wedge \sim\exists j: \text{cause\_of}_j(\sim p_{t_i})) \vee \exists j: \text{cause\_of}_j(p_{t_i})$
(4) $\forall x_1,x_2,\ldots,x_m: \Phi_a(x_1,x_2,\ldots,x_m) \supset \exists x_{m+1}: \Phi_b(x_1,x_2,\ldots,x_m,x_{m+1})$

Schema (1) and (2) describe time as a finite total ordering of points. Schema (3) makes states and their causes equivalent. This is reasonable since causes are defined to be sufficient conditions for a proposition to be true. It includes the mapping 'cause_of', expressing causality, which is written in predicate form as implications. For example, assume there are three sufficient conditions for predicate p becoming true at time t_i (written p_{t_i}). Then the three predicate calculus expressions giving those conditions are $\text{cause\_of}_1(p_{t_i})$, $\text{cause\_of}_2(p_{t_i})$, and $\text{cause\_of}_3(p_{t_i})$. We have already discussed the similarity of this aspect of our approach to circumscription in section 4.4. Schema (3) simply says that p_{t_i} is the same thing as at least one of those expressions being true or that p_{t_i} was true during the previous interval and that it did not change truth value (persistence, see chapter 7 for how this relates to the frame problem). These expressions may contain propositions, $p_{t_{i-1}}$, about the previous state (the interval $(t_{i-1},t_i]$), and propositions,e_{t_i} , about events at t_i. Schema (4) is the schema for consistency rules used by the counterfactual engine. The two expressions, Φ_a and Φ_b, are conjunctions of propositions. These are used to represent physical laws. For any given set of assertions, the counterfactual engine produces all maximal consistent subsets of that set, which may include other assertions generated by the implications in the consistency rules. All knowledge of causality is embedded within the expressions denoted by $\text{cause\_of}_j(p_{t_i})$. If any state proposition changes its truth value, the change must satisfy schema (3).

6.3.2. Version B Axioms

This is the axiom schema for the final hypothetical-based planning system. It is based on a dense timeline with a traditional < relation between time instants.

(1) $\forall t_a: (t_a < t_a) \supset \text{false}$ (irreflexive)

(2) $\forall t_a,t_b: (t_a < t_b) \wedge (t_b < t_a) \supset \text{false}$ (antisymmetric)

(3) $\forall t_a,t_b,t_c: (t_a < t_b) \wedge (t_b < t_c) \supset (t_a < t_c)$ (transitive)

(4) $\forall t_a,t_b: (t_a < t_b) \vee (t_a = t_b) \vee (t_b < t_a)$ (total order)

(5) $\forall t_a,t_c,\ t_a < t_c: \sim p_{t_a} \wedge p_{t_c} \supset \exists t_b,\ t_a < t_b \leq t_c: p_{t_b} \wedge \exists j: \text{cause\_of}_j(p_{t_b})$
(causality&density)

(6) $\exists t_a \forall t_b,\ t_a \neq t_b: t_a < t_b$ (first time)

(7) $\exists t_a \forall t_b,\ t_a \neq t_b: t_b < t_a$ (last time)

(8) $\forall t_a,t_c \sim\exists t_b,\ t_a < t_b \leq t_c: p_{t_c} \equiv (p_{t_a} \wedge \sim\exists j: \text{cause\_of}_j(\sim p_{t_b}))$
(persistence)

(9) $\forall x_1,x_2,\ldots,x_m: \Phi_a(x_1,x_2,\ldots,x_m) \supset \exists x_{m+1}: \Phi_b(x_1,x_2,\ldots,x_m,x_{m+1})$
(consistency rules)

Our inference rules are modus ponens and WIF's consistency restoration algorithm, plus the limited form of modus tollens described in Chapter 3. Most of the above axioms are self-explanatory. (1) through (4) simply define the dense, linear, totally ordered timeline of time-points. (6) and (7) state that a first and last time exist. These correspond to our initial state and goal state, respectively. (8) defines the conditions for persistence of facts into later states. (9) is the same general form for the consistency rules given in (4) of the previous section. (5) states that any change in the truth value of a fact must have a corresponding event that caused the change.

Rules (5) and (9) are WIF consistency rule schemata. The $\exists$ on the LHS is actually a limited existential over a finite domain and could be written equivalently with a universally quantified set of 'or'ed equalities. (5) and (9) are written in the $\exists$ form for the reader's convenience.

The combination of WIF and our time model produces a possible world structure equivalent to K_{ld}'s possible world structure (an irreflexive, antisymmetric, transitive, dense, total order). This provides a well understood, sound and valid semantic model for our the time model of our planning system.

6.4. Consistency, Soundness, and Completeness Results

Consistency, soundness, and completeness are important properties for logical systems. A system is consistent if it is impossible to derive a contradiction from the known rules and facts. A system is said to be sound if it produces only intended solutions and none others. A system is complete[1] if it it can always find a solution where one exists. In

[1] There is a stronger sense of completeness in which all intended models are found. We do not achieve this, because we avoid worlds with loops or other extraneous actions in them.

our particular application, consistency must refer to the initial world, the domain and system rules. Often the hypothesis is a contradiction, so these are obviously not included in the facts that are tested for consistency. The consistency of the possible worlds generated by WIF is guaranteed as long as the initial world and rules are consistent. Soundness corresponds to finding only possible worlds containing correct plans. Completeness is finding at least one world with a valid plan if any exist.

It is easy to see the advantages of these properties in a problem solving system. Consistency is necessary to ensure that the worlds containing plans correspond to actual possible worlds, since presumably all possible realities are consistent. A sound system will produce only valid plans from consistent and complete domain rules and initial world facts. A complete system will find a valid plan if there is one. Here a drawback of completeness becomes apparent: there may be a lot of possible plans, of which only a few are really reasonable.[1]

6.4.1. Consistency

The language over which WIF works (the axioms presented in 6.3.2) is decidable since it is of a form proven decidable by Bernays and Schönfinkel in 1928 [Dreben & Goldfarb, 1979]. Because we are using WIF, and assuming WIF to be complete and

[1] This is one place in which the modal categories of the facts used by WIF could have an important function, although they are not used in this fashion in the current system. An incomplete, but more useful, system is obtained by preventing WIF from throwing out certain facts in its search for consistent possible worlds. By setting the threshold so that some facts cannot be thrown out, the search is limited, at the expense of completeness. This is not a great price to pay, since humans seem to do fine without complete planning capabilities.

As an example of how we could use the modal categories in this way, consider a planner that handles trip planning, and can use various forms of transport. First we set `walk` actions at a higher modal category than `bike`, which would be higher than `drive`, and so on. All of the various forms have a distance limit to prevent walking to the next county or something. If we set the threshold very high at first and then relax it, we will find 'easier' plans first, which is a desirable result.

It may be advantageous to use a different ordering mechanism than Rescher's PMMC, so that we can choose worlds based on not only the difficulty (expressed with the modal category) but also the number of actions. This is an area that needs exploration.

consistent, the additions to the set of facts are guaranteed to be consistent in each resulting possible world. The normal definition of consistency is difficult to apply. Since we are using a consistency maintenance system, it is possible to derive contradictions within the system, although they are thrown out by WIF. If an inconsistent initial set of facts, including domain rules and the initial and final states, are given, WIF will not be able to find a plan. This is the practical view of consistency. If there is no possible world containing the initial and final states in which our system and the domain rules are true, then no plan exists.

There are two ways to go about showing consistency of a domain in our system. The first is to treat the system and the domain rules together as a set of logic statements and then to try to formally prove consistency. The other approach is more realistic and more useful in our view. First, assume the real world is consistent (if it isn't, why bother?). Then a model of the real world is consistent if all statements in the model are true in the real world. This pragmatic approach is more in keeping with the spirit of the epistemic school of counterfactual logic.

There are still questions about consistency to be answered. First of all, if the domain rules are not consistent with each other, then WIF will fail to find any worlds when those rules are used. Second, WIF assumes that the initial set of facts is consistent. Therefore, for the results to be consistent, the initial world must be consistent.

The first problem is the most difficult, and there is no general solution. Each domain must be carefully analyzed and translated into a consistent set of rules. This is true, however, of any problem solving system that contains rules about a domain. This is where the pragmatic approach discussed above is useful. While this approach is commonly used and often unsuccessful, domain rules in our system are not dependent on each other, only on the definitions of the predicates they use. This makes checking each rule much easier, since interactions between rules need not be checked. This makes our rules relatively simple to write (and analyze for consistency) compared to other problem solving systems.

The second problem can be solved very easily. The answer is to simply start with an empty world, containing no facts. This ensures the consistency of the initial world. Then WIF can be used to 'create' the initial world with its domain rules. In this manner, a simple set of basic facts can 'bootstrap' the entire initial state. For example, in blocks world, there are `over` and `clear` predicates in addition to the `on` predicate. If WIF is used to create the initial state, we need only supply the facts for the `on` predicate; the rest are derived. This greatly simplifies the process of specifying the initial state, and ensures its consistency.

6.4.2. Soundness

To show soundness for our system, we must show that our axioms are valid in all intended models and that our rules of inference preserve validity. Since the set of axioms varies with the domain, this result cannot be proven in general for all domains represented with our system. We can only assume that the domain dependent axioms (the consistency rules) are valid, and work from there. In terms of the axiom schema given in section 6.3.2, this amounts to assuming that all of the rules of form (9) are valid and that the `cause_of` relation is valid.

Our intended model is a bounded dense linear timeline, on which propositions are true or false over various time intervals. All events are instantaneous and happen at time points, and all changes in truth values are related to events by the `cause_of` relation. Next we will discuss each of the other axioms to analyze them for validity. Our axioms (1), (2), (3), and (4), expressing that linear total ordering on time points need little discussion since they are the standard axioms for this type of temporal model. Our axiom

(5) simply says that if a proposition changes truth value during some interval, then there is a point in the interval where the change occurs. This is certainly valid in all linear dense time models, even if there is an infinite sequence of events in the interval, since all the axiom says is that one point of change exists. This axiom 'creates' the points on the dense timeline necessary for events to be labeled with a time. It also enforces causality, assuming that `cause_of` is sound. Axioms (6) and (7) simply name the first and last instants on the timeline. They are valid for any linear bounded model. Axiom (8) expresses the persistence rule in our system. As long as the `cause_of` relation is correct, this will be true in all intended models since in all such models statements can only change truth value when accompanied by an appropriate event.

After this discussion of the axioms, we turn to the inference rules. One inference rule is, of course, modus ponens. The soundness of this rule is accepted without proof, as is the soundness of modus tollens. The other 'inference rule' is the algorithm used by WIF to create and destroy possible worlds from contradictions derived from the hypotheses supplied by our planner. Whether this inference rule is sound is really the question of whether WIF is sound, which is a question beyond the scope of this thesis, although we are confident that WIF, or some version of it can be proved sound.

In fact, WIF is not sound (i.e., it does not preserve validity). Construction of a sound version of WIF is very simple however. Simply use the existing system, and then take the intersection of the resulting possible worlds. This set of inferences is true in all possible worlds, and therefore valid. This would obviously restrict the usefulness of the system greatly, as very few inferences are valid in all possible worlds. Soundness is probably not a desirable property of a planning system because it limits the possible inferences too much. Systems based on Stalnaker and Lewis' counterfactuals will all suffer from this restriction since validity is the basis for the truth of a counterfactual in their logic.

6.4.3. Completeness

In order to show completeness, we need to assume that WIF is complete. The completeness of WIF is also beyond the scope of this thesis, so we will assume it can be proven, or that a provably complete version of WIF is possible. As with soundness, showing completeness is not possible in general, since each domain has its own set of axioms. We will outline how to prove completeness for a particular domain next, but first we need to summarize some important facts about our system. First, there are a finite number of predicates and named individuals in any practical (it fits on a computer) domain. Second, since agents (of which there are a finite number) can only perform a finite number of actions in any finite time interval, there is an implicit limit on the number of changes of state (which split intervals), so we have a finite number of intervals. The time model itself is dense since we can split intervals to any arbitrary depth, but the limitations of the agents in the world prevent the process from becoming infinite. Since the problem is of finite scope completeness is not an impossible goal.

The procedure for showing completeness is straightforward but tedious because of the sheer number of axioms necessary for most domains. To show completeness, pick an arbitrary causal chain of events, modeled by the causality axioms for the domain. Then show that if that chain is not found, WIF is not complete. This contradiction of the original assumption completes the proof. A technique that could be useful for this proof is the graph based formalism presented in [Joslin and Roach, 1986].

6.5. Equivalence of Temporal Model and Implementation

The tense logic system $\mathbf{K_{ld}}$ is the system embodies some of our assumptions about time. Equivalence to the modal system $\mathbf{S_5}$ has been outlined in section 6.3 and rigorously proven elsewhere [Rescher and Urquhart, 1971]. Our axioms are stronger than those of $\mathbf{K_{ld}}$ since they explicitly state irreflexivity and antisymmetry. It has been shown [Prior, 1967a] that some of these cannot even be expressed in tense logic. Thus we have a stronger system than the $\mathbf{S_5}$ and $\mathbf{K_{ld}}$ models described. Nevertheless, these systems are valuable because their theorems hold in our model, so we can learn about our system by referring to the literature on these established systems. This material is included to give some background for the possible worlds semantics of the temporal model of our system.

6.6. Time and Space Complexity

We are interested in the time and space complexity measures of our system for several reasons. First of all, the micro-worlds on which most planners are tested are a poor substitute for realistic domains, but the fact remains that our current computational abilities are not up to the task of a realistic domain. Complexity measures are one important way to determine the result of scaling up the domain without actually having to perform the programming or computer processing. Complexity measures are also important because they make objective comparisons of various algorithms possible. The third reason we are interested in the complexity measures is their importance in Chapter 7, which discusses the frame problem.

There are three aspects of this system for which complexity measures are important. First is the space complexity of the set of rules used to represent knowledge about the relationships between facts and events in the world. Second is the time and space complexity of the planning process performed by WIF. Third is the time and space complexity required to derive persistence information, since we have separated this from the direct control of WIF.

The space complexity of the consistency rule set is based on the number of predicates and the number of actions possible in the domain. First let us consider the worst case for a domain with p n-place predicates[1]. We can assume that only one clause is on the RHS of any particular rule, since we will have accounted for all rules with multiple RHS clauses by writing them in separable form as discussed in section 4.6. The maximum number of clauses on the LHS is dependent on the domain, but we will assume that any domain constraint can be written with m or fewer LHS clauses. This seems to be the case for realistic domains where only a few predicates are directly interrelated. We will also assume that the rules do not refer to specific individuals, so we need not be concerned with an unlimited number of individuals in the world. We can make this assumption because the predicates can be used to classify the individuals.[2] Then we have a maximum of m^p possible unique LHS's and p RHS's speaking purely of the predicates used for each clause. To account for the bindings between the variables in the clauses, we must multiply by $(n(p+1))!$ since each place of each predicate can be bound to any of the previous places or not at all. The result is $(n\,(p+1))!\ m^{p+1}$, or $O(p!m^p)$ which is rather large, but is unrealistically high because it assumes that the world is highly connected. Models for real domains will not need to state a relationship between every combination of LHS and RHS predicates, so we do not have to (directly) relate stock prices to the outside temperature for instance. Also, the variable binding estimate is far too high for the same reason: we do not

[1] Note that an action is represented by a predicate.

[2] If, for example, we classify all individuals into c classes (not necessarily exclusive), then we effectively avoid using any references to individuals in our rules by adding c one-place predicates and using them to 'type' the variables used in each rule.

have to bind variables that hold prices to variables that hold temperatures. Although we cannot prove it[1], we believe that adding predicates for realistic domains increases the rule set size linearly. The reason for this is the need to specify only direct relationships in the rules, and that real world concepts seem to directly relate to just a few other concepts.

The time and space complexity measures for the planning process of this system depend directly on the time and space complexity measures for WIF. Taking into account this dependency, the worst case time complexity for constructing a plan for a single goal (one WIF hypothesis) is $O(f^f)$ where f is the number of facts in the world, assuming the same p n-place predicates, with a single RHS predicate and up to m LHS predicates. This estimate assumes that every rule is applicable to the goal, and that every fact is relevant. This is the same worst case as any TMS for a single addition to its database. The space complexity of the WIF algorithm is directly proportional to the number of facts that are negated in any single possible world, $O(f)$. In the worst case, everything could be negated, but since the principle of locality seems to be applicable in realistic problem domains, we will typically negate fewer facts than the worst case. The savings is dependent on just how localized the consequences are.

The time and space complexity for persistence information can be derived directly since it involves only the PROLOG rule given in section 4.3.1 and the facts in the PROLOG database. The complexity for deriving the persistence information about one fact is $O(k^2)$ where k is the number of occurrences (at different times) of the same fact. For most cases k will be very small, usually zero or one. The number of such requests made by WIF is, in the worst case, ft, where t is the number of named time points derived in the planning process. Typically though, few facts are relevant and in any case we only derive

[1] It seems that this sort of thing cannot be proven without a proven correct representation of reality, something beyond our reach at this time.

information needed by planning; therefore, any other planning system would also have to derive it. Our time complexity is minimal, since we only compute it when needed, and we never recompute it. The space complexity for storing the persistence information is also given by ft in the worst case. These measures for persistence information are insignificant compared to the planning process complexity in any case.

7. Frame Problem

The frame problem is the name for a class of related problems including the qualification problem, the ramification problem (also called the extended prediction problem) and another problem we have identified and called the persistence problem. Since we are concerned with initial planning, we plan based on the finite knowledge that we have. In our finite models, the various pieces of the frame problem all share a common root. All have to do with the efficiency of finding a problem solution using domain knowledge that describes the interrelationships between facts about the world. They are all aspects of the question "How can I efficiently compute the results of a change in the world?". Exact definitions of these subproblems, and even of the frame problem as a whole, have never been formulated in a manner suitable for use as a basis for comparison of different approaches to the frame problem. This introduction starts with a traditional discussion of each of the frame problem's three facets. We define them unambiguously in words and give a computational definition of each aspect. The literature review uses these definitions to compare previous approaches. After the literature review, our solution to each subproblem is discussed in detail and compared to previous approaches.

Let's start our discussion of the frame problem with this classic example. When planning to go to the store, you typically do not consider whether a meteorite may have fallen on your car [McCarthy and Hayes, 1969]. On the other hand, if a meteorite does, you should be able to recognize that the car will not function properly. The difficulty lies in finding a strategy for thinking about actions that will be able to recognize the unusual conditions when they are present but not consider every unusual case that could occur when they do not apply. This is an example of the qualification problem [McCarthy, 1968], the problem of limiting the search to relevant avenues, or of limiting the qualifiers necessary for an event to be considered possible. In this example, even though a meteorite could disable your car, it is not feasible to assume explicitly that none have every time you plan to use the car.

The ramification, or extended prediction, problem is the question of how to compute efficiently the changes in the world that result from an event, both the direct and indirect effects. If the STRIPS add-delete list event representation is used for a complex domain, the space required to enumerate the cases grows exponentially [Ginsberg, 1988a] in the worst case. When calculating the results of a change, it is important to consider only relevant facts among those that may possibly change. It would be inappropriate to consider whether the price of strawberries in Taiwan changes as a result of our car getting a flat tire. Notice that the issue is not whether the price changes, but whether we even ask whether it changes. Irrelevant knowledge should not be considered by the process computing change in the world. Of course not all relevant knowledge changes; we are not claiming to find only those facts that change, just those that may potentially change.

Assuming that we are sophisticated enough not to consider irrelevant facts when changes occur, we then have no knowledge about those facts at any later times because we have only computed possible changes in the world and drew *no* conclusions about unrelated facts, not even that they stay the same. The last aspect of the frame problem basically has to do with persistence of facts about the world, so we have named it the persistence problem. It would be very inefficient to copy all of the unchanged facts into every succeeding world state, and to do so would be equivalent to considering them during the change calculations and thus improperly dealing with the ramification problem. Ideally, we would compute whether a fact is still true only when it becomes necessary to know that fact. In other words, facts *explicitly* persist only when they are relevant to the problem solving process.

The search space of the problem solving paradigm we have presented here is related to the amount of knowledge *relevant* to the goal. Since it reasons backward from the goal, we deal only with the portions of the initial state known to be relevant to the goal, thus saving computational time and storage by ignoring irrelevant information and its consequences. This, combined with consistency maintenance, is our solution to the frame problem for a complete planning system. We have achieved the minimal time complexity possible and a relatively (to other planners) low space complexity, although it is suboptimal because of the inherent tradeoff between time and space complexity. The tradeoff is explained in detail later, but in summary, facts are stored when derived in order to prevent rederivation. Thus we preserve computational efficiency at the expense of some storage space.

7.1. A Computational Definition of the Frame Problem

We would like first to present our definition in its entirety; then we will discuss it. As we have already stated, the frame problem is solely an issue of computational complexity.[1] There is no doubt about whether the correct answer in a given case can be computed, the issue is whether that computation is tractable. There are both time and space complexity aspects to the frame problem. We define each aspect, or subproblem, of the frame problem as the problem of minimizing a particular type of complexity for a particular part of the overall frame problem. We also discuss the tradeoffs between the various solutions inherent to any system that tackles the frame problem.

[1] There are cases involving infinite domains where the frame problem is not just a complexity issue. Some examples are cases with infinite consequence sets or infinite initial domains. In cases involving infinite domains of any sort, it is not clear to us how to approach the frame problem at all, since they are computationally intractable by their very nature.

There is a logical order for defining these subproblems; we start with a definition for the qualification problem. An early system in which the importance of the qualification problem became apparent was STRIPS. The problem arose in writing the event descriptions; the space to write the event descriptions exploded because of special cases involving indirect results. We define the qualification problem to be the minimization of the space required to represent the knowledge of relationships between facts and/or events. The ramification problem is the minimization of the time and space necessary to use these relationships to compute and store changes to the world after an event. The persistence problem is the minimization of the time and space complexity to derive and represent those facts irrelevant to an event that persist unchanged through the event.

These definitions may not seem intuitive at first. In particular, the qualification problem is often viewed as involving computation, whereas we have defined it solely as an issue of writing rules about the domain in a concise way. In order to understand our definition better, think about the ramification problem. All computational issues involved with using the rules are part of the ramification problem. In order to solve the qualification problem we must find a good way to write the rules that contain the domain knowledge.

Another question that can be raised is whether the persistence problem is separate from the ramification problem. In fact, it is included in the classic definition of the ramification problem. We have chosen to separate them because the problems are distinct, and a solution for one does not necessarily solve the other. Actually, there is a tradeoff involved between their solutions. One reason that this was not recognized as a separate problem to begin with was that in systems like STRIPS only one world state was stored, instead of multiple states that change over time. This is also true of the system recently developed in [Ginsberg and Smith, 1988a]. With only one state stored, there is no question of space complexity, since it is constant. There is also no memory of past states, which is a severe limitation in a realistic environment. Another drawback is that the information about past states is necessary to allow the backtracking needed for general planning [Joslin and Roach, 1986], so the information has to be stored anyway.

From these three definitions it is clear that an approach to solve one problem will influence the solutions to the other problems. For instance, the method of writing the domain rules to handle the qualification problem will strongly influence the algorithm used to attack the ramification problem.

Given these definitions it is also easy to see that it is not possible to solve all three optimally at the same time. Various systems implemented in the past have solved one or two at the expense of the others. The next section discusses these previous approaches in light of our computational definition of the frame problem.

7.2. Previous Work

The first attempts to define and solve the frame problem were made within the realm of monotonic logics: predicate calculus and first order logic among others. The situation calculus, developed by McCarthy [McCarthy, 1968], was the first attempt to handle changing situations. Using simple logical axioms of two sorts, the consequences of any action can be determined. The two types of actions were action and frame axioms; action axioms specified changes to the world, and the frame axioms specified what did not change. The number of frame axioms grew far too quickly for this solution to be used on a complex domain. In the terms of our definitions, these systems solved neither the qualification nor ramification problems. Also, since they updated one version of the world, the persistence problem was not an issue.

The next attempts made were through the use of various nonmonotonic logics. Nonmonotonic logics differ from monotonic logics in that they allow defeasible inferences.

This means that new facts can change earlier inferences even though the facts on which the inferences were based have not changed. The first nonmonotonic logic attempting to solve the frame problem was first order logic extended with the closed world assumption, or CWA [Reiter, 1978 and Genesereth and Nilsson, 1987]. The CWA adds the negation of any unprovable literal to the set of facts, based on the assumption that all true facts are stored in the database. Adding a positive literal reduces the extension resulting from the CWA, so it is nonmonotonic.

One major approach to the frame problem that employed the CWA, the STRIPS planner, represented actions by three lists of facts: a precondition list of facts that must be true for the action to be performed, a delete list of facts that become false after the action is performed, and an add list of facts that become true after the action is performed. The world was a set of facts, and applying an action to the world resulted in a new world modified by these additions and deletions. STRIPS was built based on the assumption that most actions change only a small part of the world. This approach was simple and elegant for simple domains but it became evident that for even moderately complex domains writing actions in these terms was unreasonably difficult, because every direct and indirect precondition and result had to be thought out ahead of time for *every possible case* in which the event could occur. STRIPS did have the advantage that the next world state followed explicitly from the previous world state and an action. It differed from the situation calculus in that anything that was not affected by the action persisted into the next state, and so no explicit representation of irrelevance or persistence was necessary. Each situation was a complete state description, and therefore expensive to compute and store. Using our definitions to analyze STRIPS results in the conclusion that it solved the ramification problem at the expense of the qualification problem.

Another early nonmonotonic logic developed to handle the frame problem, default logic [Reiter,1980], improved upon the situation calculus by replacing the many frame axioms with a single general frame axiom which basically stated that a fact persists through an action if it is consistent to do so. This still suffers from the drawback that each fact in the world must be checked for consistency in order for them to persist through the action, which is an intractable problem. Default logic was developed to solve the qualification problem by allowing many of the frame axioms from the situation calculus to be written as one or a few default rules. Applied properly, it does solve the qualification problem. Since it was designed to allow unchanged facts to persist without writing all the necessary frame axioms, it is not relevant to the ramification problem but it is relevant to the persistence problem, which it does not solve.

Around the same time as default logic, several other nonmonotonic approaches to the frame problem were proposed, of which circumscription [McCarthy, 1980] was the major one. Circumscription is best described as an extension of the closed world assumption (if a fact is not known to be true, then it is assumed to be false) to rules. Circumscription is the assumption that all rules affecting a literal are known. Then it is possible to infer such things as $c \supset a \vee b$ from $a \supset c$ and $b \supset c$ (where these are the only two axioms with c on the RHS), which is clearly not valid without the circumscription assumption.[1] Circumscription concentrated on solving the qualification problem, essentially by requiring that all qualifiers be explicitly stated.

Another difficulty with circumscription is a class of planning problems characterized by the example known as the Hanks-McDermott problem, also called the Yale Shooting Problem. There are two minimal models for the specification of the problem

[1] Circumscription operates on predicates also, not just literals as in this example.

according to circumscription, where there is clearly one correct answer. We will discuss the problem and our system's solution of it later in this chapter. In [Lifschitz, 1986], a variant of circumscription called pointwise circumscription was developed that appears to handle this difficulty, although it is not fully developed as yet.

Hintikka, in an attempt to improve on circumscription, has developed a method he calls model minimization [Hintikka, 1988]. He discusses several problems with McCarthy's version of circumscription; among them are the fact that it does not even restrict models to minimal ones in all cases, and that it can produce inferences that are too strong. His model minimization solves these difficulties, and it also allows infinite models to be minimized, unlike circumscription. It is not clear whether it resolves the difficulty raised by the Hanks-McDermott problem. We are also not sure what the time and space complexity measures of an implementation of this formalism would be and so cannot determine to what extent the frame subproblems have been solved.

Another approach that has some similarity to our own, Kowalski and Sergot's Event Calculus [Kowalski and Sergot, 1985], claims to avoid the frame problem altogether. A close examination of their work reveals that they have solved the qualification problem with what is essentially Reiter's default logic. Their improvement lies in the fact that they handle the persistence problem as well. The drawback is that they cannot handle contradictory information at all, since they are working in a first order logic framework. This means that they can handle information that changes over time, but that they can never correct incorrect information. This ability is important to any system that must handle incomplete information about a changing world.

A very early attack on the frame problem with consistency maintenance was presented by Patrick Hayes [Hayes, 1973]. His solution strongly resembles later consistency based approaches. He discussed a consistency maintenance system developed at Stanford, although it was never formally presented in the open literature, so it is difficult to analyze here.

Ginsberg and Smith present a new approach they call the possible worlds approach [Ginsberg and Smith, 1988a and 1988b]. They also employ consistency maintenance techniques to model change, although in a more specific way. They also claim to have solved the frame problem. Their algorithm for computing change is based on the counterfactual logic of Stalnaker and Lewis, although there seem to be some discrepancies in this regard.[1] They seem to be restricted by a discrete model for time. A major drawback of their system is that they keep a single world model, and change it when a change occurs in the world modeled. This means that no information about the past is kept for later use; it is thrown away as soon as it changes.

In [Schwind, 1987], Camilla Schwind defined an 'action logic' for describing time and change, and reduced the frame problem to the problem of consistency proving. Unfortunately, it also suffers from the restriction of a single world that is updated to reflect change.

In [Shoham, 1988] Yoav Shoham defines and discusses a temporal logic called the logic of chronological ignorance. He also claims to have solved both the qualification and ramification problems with his system. Unfortunately, it is also not very clear how to apply it to planning problems, since it was designed for, and is limited to, prediction applications, according to Shoham.

[1] See section 7.3.2 for the discrepancies.

7.3. The Qualification Problem

This section explains how our system solves the qualification problem. The qualification problem is the difficulty encountered in writing knowledge about relationships between facts so that irrelevant facts about the world are not needed in the process of reasoning using the relationships. It has already been shown that planning (in a formally complete sense) cannot be performed without search [Joslin and Roach, 1986]. There must be a compromise between two conflicting demands; we want to avoid considering irrelevant cases but we must recognize all relevant cases. This entails some sort of explicit relationships between classes of facts so that a determination of relevance can be made. In our system indirect relationships are implicitly defined by chains of explicit relationships. This is also true of any system that does not represent all consequences of a change explicitly as does STRIPS or the situation calculus.

The relationships are expressed in our system with consistency rules. Transitivity applied to these rules implicitly expresses the indirect relationships. By using consistency rules, we have reduced this information to the bare minimum. We need only state the direct consequences of an action, and other rules expressing constraints on the domain (such as no two objects may be in the same place at the same time). These enable WIF to disallow impossible actions, and derive all the direct and indirect consequences of an action. For simple domains, both the older style systems and our system have roughly the same number of rules, but for larger systems, our method needs a set of rules that is both simpler and smaller.

The search process in our system is handled by our hypothetical reasoning engine, WIF, as part of the consistency maintenance process. Given a literal representing some new piece of knowledge, WIF can directly look up all consistency rules containing that particular literal on their LHS. The rule(s) are then instantiated with that literal and thus any other literal in the instantiated rule(s) may also be relevant to the new literal. The transitive closure of this process finds all relevant literals and can therefore determine what parts of the world model are relevant to the new knowledge. Some of the relevant literals are new facts that have been inferred about the world. Others contradict previous knowledge and therefore signify changes in the world. Refer back to Chapter 3 for a more complete explanation of the operation of WIF.

There may be some concern about the cost of performing the transitive closure of the operation described above. It seems possible that this could explode to include the entire world very easily. While this is a valid concern, it will only happen in domains where every fact is affected by every other. If the domain obeys the principle of locality, the set of relevant facts found by the above process will also obey the same principle. Realistic domains tend to obey the principle of locality very strongly. Except in a few cases such as nuclear wars, the effects of any action are very localized.

7.3.1. The Potato in the Tailpipe Problem

In this section, we consider a typical example of the qualification problem and show how our system handles it. Instead of the meteorite problem discussed in the introduction to this chapter, let us consider a potato in the car's tailpipe. This is not something we normally consider when planning to go to the store. On the other hand, most people can readily understand the consequences of such a fact: the car will not start. Following are consistency rules to model the relevant portions of a domain where this could occur. This example illustrates how our system can draw the correct conclusions from the knowledge of a potato in the car's tailpipe without explicitly assuming that nothing is in the tailpipe every time we use the car.

```
(start (in tailpipe *obj) *t)
                    -> (start (blocked exhaust) *t)
```

```
(start (blocked exhaust) *t)
                    -> (stop (function engine) *t)

(stop (function engine) *t)
                    -> (stop (function car) *t)

(occur (drive car) *t)
                    -> (state (function car) *t)
                       (state (have carkeys) *t)
                       (state (in car) *t)
```

Adding an occurrence of `(drive car)` to the above rules causes WIF to make three explicit assumptions: `(function car)`, `(have carkeys)`, and `(in car)`. The last two of these will (hopefully) produce other inferences such as facts about moving to that car and getting the car keys. This is possible because `(have carkeys)` and `(at car)` are on the LHS of other consistency rules not stated here.

As discussed later in this section, modus tollens is not applied by WIF to some rules as part of our model minimization that eliminates unwanted inferences. So `(function car)` does not cause `(not (blocked exhaust))` or `(not (in tailpipe *obj))` to be inferred. From the contradiction that follows from adding `(in tailpipe potato)`[1] and its consequences to the above rules with `(usable car)` and its consequences, we can see that this is a nonmonotonic system. There are two possible resolutions to this contradiction. We can either conclude that the car is not functioning or that the potato is no longer in the tailpipe (presumably the exhaust gases blew it out when the car was started). Notice that we have no way of deciding which is the 'correct' resolution to this action. This is to be expected because we have not modeled the forces within the exhaust system. Ginsberg and Smith state that this inability to distinguish

[1] This is where this section is relevant to section 5.5 on the planning algorithm. Consider a case where our agent has planned to go out and drive the car. He walks out to the car and notices that there is a potato in the tailpipe. Since he is using WIF to maintain the consistency of his beliefs, he realizes that the exhaust is blocked, that the engine may not function, that the car may not function, and that he therefore may not be able to drive as he had planned.

which resolution is correct is a shortcoming of certain earlier systems (and presumably ours as well) [Ginsberg and Smith, 1988b]. Their specific objection is that there is no distinction made between facts that may change as a result of an action and facts that can prevent an action from occurring.

We disagree, because either resolution could be correct and without a model sufficiently detailed to decide, either answer would be incorrect in some cases, and therefore not a correct inference. If the domain modeler wants to decide one way or the other, the consistency rules for the domain can must be written in such a way as to make the desired answer the default. Ginsberg and Smith's answer is to state specifically which facts can prevent an action and which facts can change as the result of an action. We consider this approach a poor one because it forces detailed and explicit decisions about the domain to be made by the domain modeler where general rules should suffice to let the system throw out cases that are not correct. For example, to decide whether a potato should disable the action of starting the car or be changed by it should be decided by some rules in the system that state the force of the exhaust gases and the tightness of the potato. This is more general and also correct; their approach would have to be wrong in at least one of the two following cases. In the first, the potato is very loose in the tailpipe. In the second, the potato is very, very, very tight in the tailpipe. Obviously, a decision ahead of time either direction will be wrong in one of these two cases.

Realize that this is only an example set of rules for the problem; in the interest of brevity they have been simplified. A more realistic, generalized, version is certainly possible, and probably more useful in the long run because while a more general system of rules requires more inferences to come to a particular conclusion, it is much easier to add new information to the system, and less of it is in the form of special cases. For instance in the above example, we have a general rule about what can block the tailpipe, but not about what parts of the exhaust system can be blocked. The appropriate level of detail supported in any particular implementation depends on the domain, but in order for the system to be adaptable and expandable, a general approach as opposed to a special case approach is better, even if the special case approach is initially easier to write.

The ability to suppress the application of modus tollens has some interesting implications for our system. We have already seen that it allows us to write our consistency rules so that unlikely inferences are ignored, although they are available when needed. In effect, this is our model minimization technique. The result is that the unwanted inferences are minimized out of the possible worlds generated by our system. Whether this occurs depends on the domain and causality rules for the particular domain.

Rules that are written so as to avoid modus tollens can allow inconsistent worlds to be created. For instance, a simple case of this effect can be seen in a world with the rule $a \supset b$. If ~b is added, and modus tollens is suppressed, then both a and ~b will be allowed to stand, despite the rule to the contrary. This is because only new facts are checked against the LHS of consistency rules. While this can produce inconsistent worlds if applied carelessly, it is a very useful tool for solving the qualification problem by eliminating certain unwanted inferences. The production of inconsistent worlds as mentioned above can also be prevented by using the 'bootstrap' procedure mentioned in 6.4.1. If we follow this process, then any world with a will also contain b, so ~b will produce an explicit contradiction with the database instead of being allowed to pass as a consistent addition.

7.3.2. Stalnaker's Counterfactual Fallacies

At this point it seems appropriate to bring up some famous examples first proposed by Stalnaker [Stalnaker, 1968] and discussed at length by Lewis [Lewis, 1973], two major figures in the counterfactual logic field. Lewis discusses three inference forms that he claims are not valid inference forms for hypotheticals, although they are valid in monotonic logics. He calls the three fallacies he identified the *fallacy of transitivity*, the *fallacy of strengthening the antecedent*, and the *fallacy of contraposition*. We will discuss the first, transitivity, as an example; the other cases are similar. Remember that a counterfactual is in the form of an inference,

p -> q

with the English translation of 'If it were the case that P then Q also'.

One of his examples of the fallacy of transitivity is the following case:

If J. Edgar Hoover had been born a Russian, the he would have been a communist.

If he had been a Communist, he would have been a traitor.

∴ *If he had been born a Russian, he would have been a traitor.*

This seems like a valid case of transitivity, the first two statements seem true, and the last seems false. He claims that the first two are *valid* inferences, that is, true in all *nearest* possible worlds, while the second is not valid. His definition of nearest possible world is somewhat fuzzy, but let us grant him that the first two are true in all of them. Since the conclusion is not, transitivity would seem to be invalid, a sad state of affairs.

We would now like to analyze this example in different terms. His definition of validity over only nearest possible worlds is what gives the difficulty. Another, and we think correct, way to examine this problem, is by not ignoring the qualification problem. For example, the second statement is true *with the qualification* that Hoover was actually an important American politician. This is the assumption that Lewis implicitly makes when he speaks of nearest possible worlds. Both of the first two hold numerous such implied assumptions. Any computational method for implementing Lewis' counterfactual theory will have to deal with these qualifications. We want to avoid both the ambiguity of Lewis' definition of nearest possible worlds and the fallacy of transitivity by axiomatizing such examples correctly and solving the qualification problem.

The problem with this example is that the first two inferences are not valid in general. Being a communist does not make one a traitor. Being a communist makes one act in the interests of communism. If one were also an important American politician, where America's interests are different from those of communism, then one would deliberately perform actions against the interest of one's country. This is the definition of being a traitor (or close enough). We, of course, have not fully represented the intricacies of politics in these few statements, but we believe that an approach like this, to the necessary level of detail, is more appropriate than throwing away a valuable tool such as transitivity. Instead of questioning the validity of the inference rules because they are invalid in Lewis' system, we would question his system for not allowing such intuitively correct and useful forms to be used. We are trying to develop a system that can deal with practical issues. We must start with the pragmatic considerations of what inference forms are useful and intuitive, and then base our logic on them, rather than decide what forms are valid based on an arbitrarily defined logical system.

Ginsberg and Smith describe a system in [Ginsberg and Smith, 1988a and 1988b] that they state is based on the counterfactual logic of Stalnaker and Lewis. Their system seems to have a major flaw in that it allows these inference forms that are not valid in the logical system they claim to be using. Transitivity is not valid for Stalnaker and Lewis' counterfactuals because they insist that all inferences be valid so Ginsberg and Smith's system either cannot allow transitivity of counterfactuals, or is not true to the counterfactual logic of Stalnaker and Lewis. Ginsberg and Smith do not state that they are not following Lewis' logic, but they seem to be allowing transitivity since their examples use it.

7.4. The Ramification Problem

WIF also provides an elegant and efficient solution to the ramification problem [Roach, Eichelman, and Whitehead, 1985] [Graham, 1986], which has also been called the extended prediction problem. Given a literal representing a new piece of knowledge, WIF uses consistency rules to derive the direct consequences of the new knowledge. These consequences are also new knowledge and so all of the consequences of a change, both direct and indirect, are determined using this simple procedure, applied recursively. There is no need to forsee all possible interactions between various facts as is required by add-delete list representations of change. All that is required are the direct, and therefore more limited and easier to discern, relationships between facts in the domain. An event is a fact just as a state is a fact, so the consistency rules are the means for expressing both causality and physical constraints in the domain. An example of a causal rule is

```
(occur (pickup *person *obj) *t)
(state (at *obj *loc) *t)
                    -> (start (holding *person *obj) *t)
```

while an example of a physical constraint is

```
(start (at *obj1 *loc) *t)
(state (at *obj2 *loc) *t)
                    -> (= *obj1 *obj2).
```

The first rule would interact with the next two rules to change the location of an object being held because if the person holding the object moved, the first rule would give the object a new location, and the second would make the old location invalid.

```
(start (at *person *loc) *t)
(state (holding *person *obj) *t)
                        -> (start (at *obj *loc) *t)

(start (at *obj *loc1) *t)
(state (at *obj *loc2) *t)
                        -> (= *loc1 *loc2)
```

This shows how our methodology handles the ramification problem. Only the intuitive direct consequences of an action must be specified. Other consequences are derived from them, including changes in previously believed knowledge.

Another advantage of this method that can be seen here is that to add new constraints between facts, one need only add the rules expressing those constraints directly; other rules, as long as they are still true, need not be altered. Even if a change is necessary, the alteration should be limited to the one or few rules that directly express the constraint to be changed, a consequence of the principle of locality. This is a great advantage because it makes additions and changes much easier and more reliable in the same sort of way that locality of scope in programming makes changes easier and more reliable.

7.4.1. A Robot Carrying a Mop

One of the early manifestations of the ramification problem was in a world known as JANITOR [Roach, 1972 and Siklossy and Dreussi, 1973]. The problem concerned a robot janitor that could pick things up and carry them around. For example, he could be carrying a mop, or a bucket, or nothing. Early implementations required a rule for the `move` action for every combination of objects that could be carried. Later improvements led to systems with one rule for each object, stating that the object moved if it were carried. Default logic approaches can accomplish this with only one rule, but pay a steep price in that they have to test every known fact to see if it changes. Our system handles this case with only three rules, for any number of objects of any type that may be carried, without looking at anything but the relevant facts. These three rules move the robot, and everything it is carrying, and remove them from where they were before, without checking every object in the world to see if it is being carried. They are very intuitive, simple, rules that are easy to express. Here they are:

```
(occur (move *rob *loc) *t)
                      ->    (start (at *rob *loc) *t)

(start (at *rob *loc) *t)
(state (hold *rob *obj) *t)
                      ->    (start (at *obj *loc) *t)

(start (at *thing *loc1) *t)
(state (at *thing *loc2) *t)
(<> *loc1 *loc2)
                      ->    (stop (at *thing *loc2) *t)
```

The first rules states that if a robot moves, then it is at the destination afterwards. The second rule says that whatever it holds is also at that location. The third says that anything that moves is no longer where it originated from. WIF can correctly derive the consequences of a `move` action for the robot and everything it is carrying using just these rules.

7.4.2. The Yale Shooting Problem

Another example that has given many contemporary planning systems some difficulty is the Yale Shooting Problem. The world has three actions, `load`, `wait` and `shoot`, and two predicates, `loaded` and `alive`. In the initial state the gun is unloaded and Fred is alive. For the sequence of actions `(load gun)`, `(wait)`, `(shoot Fred)`, the intended result is, of course, `(not (alive Fred))`, but circumscription produces two minimal models [Hanks and McDermott, 1987], the intended one and one in which the gun is mysteriously unloaded during the `(wait)`. In our system the domain would be modeled as follows.[1]

```
(state (not (loaded gun)) t0)
(state (alive Fred) t0)

(occur (load gun) *t)
                         -> (start (loaded gun) *t)

(occur (shoot Fred) *t)
                         -> (stop (loaded gun) *t)
                            (stop (alive Fred) *t)
```

Then the sequence of actions `(load gun)`, `(wait)`, `(shoot Fred)` results in a single final state where Fred is dead, as intended. The reason that we have only one model is that we do not attempt to minimize the size of the set of changes for a particular predicate as in circumscription; we just calculate the changes entailed by facts that are known. Our minimization process minimizes inference rules to solve the qualification problem, instead of minimizing changes to the world, as in circumscription. We do not make unnecessary assumptions about the world during the reasoning process. There certainly is no knowledge of anyone unloading the gun while we are waiting, therefore 'the thought never occurs' in our planning system. Our approach is more like default logic than circumscription in this regard. As will be shown later, we improve on the default reasoning approach when handling the ramification and persistence problems.

7.5. The Persistence Problem

The third subproblem of the frame problem is the persistence problem. It concerns the efficient derivation and storage of facts not relevant to events. Starting with the issue of

[1]We include an explicit `wait` action here in order to keep the problem the same for comparative purposes. It is not necessary, as has already been discussed in section 4.4, and as can be seen from the lack of rules involving the `wait` action.

space complexity, it is unacceptable to copy every fact into each new world state. Systems that derive and copy all facts that persist unchanged through an action waste both time and space. Previous systems have used this method, or other more sophisticated methods such as storing only changes at each succeeding time interval.

An optimal system for minimizing fact storage requirements would store *at most* only those facts known or assumed to be true (or false since we do not use the CWA) that are otherwise underivable. These underivable facts are those that are directly perceived and also those facts stating any assumptions made about the world. It is important to store all directly perceived facts because they are the basis for beliefs about the external world; perceptions are not redundant, so any perception may become useful at a later time. They must be stored because even if the information is otherwise derivable (or consistently assumable), the fact that it was a direct perception is not derivable.[1]

Assumptions must be explicitly stored in order to recognize when an assumption is incorrect. Since knowledge derived for use in planning is either logically derivable (hence redundant) or is uncertain but assumed based on other knowledge (persistence is a good example of this), any knowledge derived during planning falls into one of these two categories.

---

[1] As an example, consider the following case. You see me walk into a room with no other doors or windows and lock the door behind me. The door has a window in it and you watch me the entire time I am in the room. As a planning system, you might be tempted to throw away the perceptions of me through the window since they follow from your other knowledge. Later that day, the police call and ask where I was while the bank next door was being robbed. You say that I was in a locked room the entire time. They then tell you of the secret tunnel from that room that you didn't know about. If you had forgotten the perceptions of me in the room, and just relied on the now-invalid derivations of my presence in the room, I no longer have an alibi. Luckily for me, however, you remember *seeing* me in the room during the time of the robbery and can give an alibi.

Perhaps if the perception were monotonically derivable from other perceptions and also above suspicion it need not be stored. This would require a monotonic logical mechanism in WIF, which can be achieved by making all facts undoubtable. Even if it were performed, it does not seem that it would produce a useful saving in space, especially in a nonmonotonic system where the agent has incomplete knowledge. Then very few perceived facts could be also derived monotonically and therefore be redundant. Since these conditions seem likely for any useful planning system, this small possible gain does not seem worth the effort.

The semantics of the rules in our approach is similar in some ways to the semantics of default rules. We differ markedly from default reasoning, however, in the computation required to discern the results of an event. In order to force facts to persist unchanged into successive world states, default reasoning systems use a rule of the form

$$p_t : Mp_{t'} \ / \ p_{t'}$$

which, translated, says that if it is consistent to let fact p persist, then do so. Literally, it says that if we know p at time t, then assume p is true at t' if it is consistent to do so. This test is performed on every fact in the world. We, on the other hand, only calculate changes to the world. Persistence is calculated only on an as needed basis, saving the effort and space required to explicitly force every unchanged fact to persist.

In addition to the minimal set of facts stored by the optimal system discussed above, we also store facts derived during planning. These additional facts are stored simply in order to avoid rederiving them again later should they become useful. Since they have been derived once already, they are known to be relevant to the planning process, as discussed earlier in 7.2 and 7.3. This method is suboptimal as far as space complexity goes, but necessary in order to achieve optimal time complexity, which is a more important objective in our view.

Our system definitely stores less than those that copy every fact into each state. We store all of the changes, both direct and indirect, that we derive. Some other systems are built in this fashion. There are systems such as the situation calculus that can be implemented so that they store only the initial state and the events. This comes at the expense of rederiving every fact every time it is needed. We consider our solution a happy compromise. We have reduced the space requirements as far as possible without increasing the processing required.

In any search algorithm, there is tradeoff between processing and storage requirements. This method manages to keep a respectable space complexity without sacrificing processing speed. When a fact is needed, it is derived with a processing cost similar to the cost of deriving it in any other system. The difference is that only useful facts[1] are derived and stored, saving both time and space. In order to prevent re-derivation, if a derivation fails the negation of the fact is stored, if that can be derived. Otherwise, multiple possible worlds are created to represent the uncertainty of the fact. This is actually the *open* world assumption, something that has not ever been implemented in a planning system as far as we know. The open world assumption is actually just the assumption that anything underivable is unknown [Reiter, 1978]. Multiple possible worlds is our method for handling the unknown facts. It allows us to plan in spite of the uncertainty, by creating alternative worlds where things are certain. The alternative worlds created to handle uncertainty are not as storage-intensive as they would seem to be, because WIF stores them in a difference tree format. It is also important to remember that only relevant unknown facts initiate the splitting into alternatives. While unfortunate, it seems inevitable that planning in spite of a lack of relevant information will be expensive in any case, unless we resort to guessing.

One concern that could be raised here is that the storage of multiple worlds could explode the space requirements of our system. While it is not possible to store them for free, WIF stores the worlds in a tree form, effectively allowing shared storage of common facts. Also, since WIF searches in a depth-first fashion, worlds that turn out not to be viable are discarded. Therefore, using the tree representation, at most only consistent possible worlds plus one world under investigation is ever stored at a time. Another answer to the concern about the storage space for multiple worlds is that our solution

[1] Useful in the sense that they have been asked for during planning

method is just as viable for multiple worlds as for one, and will compare favorably for the same reasons discussed above to other methods storing multiple worlds.

7.5.1. Strawberry Prices in China

As an example of our system's method of handling the persistence problem, consider whether the price of strawberries in China should change because of my trip to the 7-Eleven. Even if the system knows about strawberries from China, it will never 'think' about them because only relevant knowledge is mentioned in the consistency rules. Since the system assumes its original database is consistent and complete, and only checks new facts, without any linkage between the trip and strawberry prices in China, consideration of my trip to the 7-Eleven will not 'fire' any rules having to do with the prices in China. Now that we have determined that the price is not changed, or even checked for change, how do we derive it later when we need it?

Looking back to section 4.3, there is a rule given for computing persistence. Facts persist whenever they are *askedfor*[1] and it is consistent for them to do so. So, whenever the next time we need to know the price it can be computed in this way. If we a never ask (a likely course of events), it will never be computed or stored, and we will save the time and space needed for the derivation.

We will give only this one example for the persistence problem since we think that it clarifies how *and when* our system computes persistence information.

7.6. Efficiency Discussion

Now that we have discussed both the definition and our approach to the three subproblems of the frame problem in detail, we would like to analyze the computational

[1] Because that rule is backward chaining

complexity our system for each subproblem. A direct comparison to other systems is difficult for several reasons. First, systems placing a different priority on various areas will be better in some areas and worse in others. For example, STRIPS has a smaller execution time complexity, at the expense of its exponential rule space complexity (and the difficulty of writing the rules correctly). Therefore we start by talking about the rationale behind our choice of priorities.

We feel our choice is clearly the best for one very simple reason. The difficulty of writing the rules correctly must be limited because this task is still performed by humans, and we cannot grasp the complexity of a large system all at once. If it is possible to describe each predicate in terms of only a fixed number of other predicates, then the number of rules is linearly proportional to the number of predicates. A more complex set of relationships will grow too quickly for us to scale it up to a useful problem domain.

Our analysis of the frame problem did separate the derivation of change and persistence into separate areas, and one of our major performance gains is in the area of persistence. As discussed, we only derive and store persistence information when it is needed. A purely forward chaining system cannot do this, so we used PROLOG instead of WIF for this task.

In order to compare our complexity measures to other planning systems, we also need their complexity values. Unfortunately, this has not been done in a formal manner for most other planning systems. In [Ginsberg and Smith, 1988a and 1988b], the single action prediction complexity of some paradigms are examined, and it is shown that non-exponential algorithms for single action prediction exist.

The second difficulty with attempting to compare planning systems, is that for any complete planning system, the worst case time complexity is always at least exponential

[Joslin and Roach, 1986]. Since current systems such as ours and others are exponential, we cannot expect any improvement in the worst case performance for planning systems.

This leaves us with comparisons of average-case problems, which leads to the third difficulty. We have no idea what an 'average' planning problem looks like, much less how to characterize it formally for further analysis, so any comparison other than worst case is suspect. All we can do is discuss the qualitative differences between the approaches.

As we said, for any planning system with the same priorities as ours, the worst case time complexity is exponential. There is really no way around this, as there is no general way to determine ahead of time which paths will be useful and which are not [Joslin and Roach, 1986]. Of course, there are many possible specific optimizations that could be performed in order to prune the search tree. These may be applicable in enough cases to make the exponential nature of the problem irrelevant. It may also be possible to use some sort of learning system to look for optimizations automatically by trying large numbers of simple plans.

8. Results and Conclusions

We start this chapter with several examples of our results on several important planning problems. We then discuss the contributions of this research to the planning field. The next section focuses instead on the unanswered questions, some of which our research has raised, and some of which seem approachable with new insights from a hypothetical reasoning perspective. The chapter, and this thesis, closes with a short conclusion.

8.1. Some Classic Planning Problems

In this section we would like to give the results of our system when applied to some classic problems in the planning field. There are four major classes of planning problems. The first class is the class of linear problems. They can be solved with simple goal ordering techniques. We do not give an example of this type of problem, since it is so simple to solve, and shows nothing interesting about the operation of our system. We do include the domain rules, initial state, goal, and resulting plans for a nonlinear problem and an indirect goal problem, along with a short discussion about the interesting aspects of each. The process by which WIF reached these results was discussed in detail in section 5.4.

8.1.1. The Blocks World Nonlinear CAB Problem

Our system solves this nonlinear problem in a very straightforward manner. From a commonsense description of the world, we deduce more and more about the nature of the world in which the goals are achieved, until finally the plan is complete.

Domain Rules

```
(state (on *a *b) *t)
(persist (not (state (on *a *b) *tprev)) *t)
(earlier *tprev *t)
     ->   (during *tprev *tnew *t *before *after)
          (earlier *before *tnew)
          (earlier *tnew *after)
          (start (on *a *b) *tnew)

(not (state (on *a *b) *t))
(persist (state (on *a *b) *tprev) *t)
(earlier *tprev *t)
     ->   (during *tprev *tnew *t *before *after)
          (earlier *before *tnew)
          (earlier *tnew *after)
          (stop (on *a *b) *tnew)

(start (on *a *b) *t)
(not (occur (move *a *b) *t))
     ->   (inconsistent)

(stop (on *a *b) *t)
(not (occur (move *a *place) *t))
     ->   (inconsistent)

(occur (move *a *b) *t)
(persist (state (on *any *a) *t) *told)
     ->   (inconsistent)

(occur (move *a *b) *t)
(persist (state (on *any *b) *t) *told)
     ->   (inconsistent)

(state (on *a *b) *t)
     -> (state (over *a *b) *t)

(state (over *a *b) *t)(state (over *b a) *t)
     -> (inconsistent)
```

Initial State {t0}

```
(state (on c a) t0)
(state (on a table) t0)
(state (on b table) t0)
```

Goal State {t0,t1}

```
(state (on a b) t1)
```

```
(state (on b c) t1)
```

Result World {t0,tb,tc,ta,t1}

```
(state (on c a) t0)
(state (on a table) t0)
(state (on b table) t0)
(stop (on c a) tb)
(occur (move c table) tb)
(start (on b c) tc)
(occur (move b c) tc)
(start (on a b) ta)
(occur (move a b) ta)
(state (on a b) t1)
(state (on b c) t1)
```

8.1.2. An Indirect Goal Problem in Blocks World

In this problem we find that we have a goal, `over`, for which no rules describing causes exist. There is no association of an action with the `over` state in any rule. Previous planning systems have had to directly associate solvable goal predicates with some action, usually by introducing a new operator. Once again, we see the leverage that our world description achieves. We can describe new predicates, such as `over`, without having to artificially associate them with actions. We need only to describe `over` in terms of `on`, which is a very intuitive and logical description of `over`.

Additional Domain Rules

```
(state (over *a *b) *t)
(not (state (on *a *b) *t))
     ->   (isa thing *place)
          (state (on *a *place) t1)
          (state (over *place *b) t1)
```

Initial State {t0}

```
(state (on a b) t0)
(state (on b table) t0)
(state (on c d) t0)
(state (on d table) t0)
```

Goal State {t0,t1}

```
(state (over a d) t1)
(state (over c b) t1)
```

Result World `{t0,tb,ta,tc,t1}`

```
(state (on a b) t0)
(state (on b table) t0)
(state (on c d) t0)
(state (on d table) t0)

(start (on c table) tb)
(occur (move c table) tb)

(start (over a d) ta)
(start (on a d) ta)
(occur (move a d) ta)

(start (over c b) tc)
(start (on c b) tc)
(occur (move c b) tc)

(state (over a d) t1)
(state (on a d) t1)
(state (over c b) t1)
(state (on c b) t1)
```

8.2. Contributions of This Thesis

The primary aim of this research was to construct a planner based on hypothetical reasoning driven by a commonsense causal theory. On this basis alone, we were very successful. Our planner can solve all types of problems extant in the literature, including nonlinear problems and the recently identified Yale Shooting Problem, and it is relatively easy to construct a planner for a particular domain. The code required to implement a planner for a domain using our system is simply a set of WIF consistency rules, specific for the domain. The system includes all general rules and PROLOG code for deriving persistence and creating new points on the timeline.

Our representation scheme includes an 'isa' class hierarchy, but instead of mutually exclusive classes, which can force very limited and artificial implementations, members of our classes inherit particular values of properties from the class. The mutual exclusion is then achieved with one simple rule stating that an individual can only have one value for any property.

Our domain descriptions are in the form of intuitive rules. Only the direct consequences of an event or direct relationships between facts need to be specified; indirect relationships are derived from the direct ones. Because of this, rules in our system can follow the principle of locality to the same extent as the domain being modeled does, and for realistic domains, effects and relationships are typically very localized. This makes creation and modification of the rule set much simpler and more reliable. Extensions are reduced to additions of rules. Irrelevant rules do not have to be changed at all.

We have identified a new type of planning problem, the indirect goal problem, that we can also solve, where previous systems cannot. These problems are characterized by a goal statement that is not the direct result of any action. Since previous systems looked for actions to solve a goal by examining the results of actions, they could not solve this type of problem. Our system can derive other information about the goal state from the statement of the indirect goal, thereby finding information about the goal state that can be directly achieved by an action.

We have approached this research with a fresh look at the basis of planning systems: time, change, and causality. We analyzed these topics in terms of the needs of problem solving systems. The result of this analysis was a set of assumptions about time, change, and causality that we consider accurate for 'real-world' domains. Because of the care that was taken to provide as general a model as possible for representing time and change, the resulting system is an excellent basis on which other, more complex and/or more specialized systems can be built. Two useful examples of possible extensions are interval based events, and a continuous time model. The complete set of assumptions on which we based our system are these:

- Time points map onto the rational number line
- The ordering on time points is a total order
- States are homogeneous (left-open) time intervals, over which propositions are true or false
- Events are instantaneous, and are in fact just state transitions
- A *cause* for a predicate is a sufficient condition for that predicate to become true, or to become false in the case of a negated predicate
- Every change (in the truth value of a predicate) in the world has a cause

With these assumptions, we have defined the behavior of our system. There were string resemblances to default reasoning in our persistence derivation technique. There was also a similarity to circumscription, in the technique for writing causality rules. We think that we have managed to combine the favorable aspects of these two approaches, in places where they are useful, while avoiding some of their difficulties, such as the difficulty circumscription has with the Yale Shooting Problem.

We characterized our model of time with the tense logic axiom system $\mathbf{K}_{ld}$ and the S_5 modal system. Using an axiom schema representing the facts and rules in our system, we have shown that consistent, complete, and sound implementations can be created with this approach, as long as WIF itself is consistent, complete, and sound. We showed that our system modeled time and events in a manner that was equivalent to Turner's system, an intuitively correct temporal logic taking events to be primary.

On the other hand, we have also shown that it can be advantageous to minimize the models by not applying modus tollens to certain rules. While this sacrifices completeness and soundness, it allows the qualification problem to be solved very elegantly. It seems likely that it is not possible to solve the qualification problem and have completeness at the same time. Fortunately, the qualifications that we minimize out of the model are not useful; that is why they are considered a problem if derived.

The qualification problem is only one of the three aspects of the frame problem that we have discussed; the other two were the ramification and the persistence problems. We are the first to define the frame problem precisely at all, to our knowledge. Furthermore, our definition is one based on the time and space complexity of a solution. With our definition, various approaches can be objectively compared, which also was not possible before. One of the three aspects of the frame problem that we defined was the persistence problem. Up until now, it has been considered part of the qualification problem, but we argued that it should be considered separately, since it can be solved separately.

We discussed the reasons for the lack of a single optimal solution to the frame problem. Each of the three subproblems are interrelated, and the solution to one affects the other two. We then discussed the priority we placed on each of the three problems, and discussed our solution.

We seem to have solved the persistence problem optimally, only deriving and storing information about states that persist through change when the information is needed, and never rederiving it. Our solution to the qualification problem is based on the technique of minimizing the model by not applying modus tollens in some cases. This method allows our system to avoid making useless inferences, while still knowing about them in the unlikely case that they become useful. Our solution to the ramification problem lets WIF use the domain rules to derive the consequences of events, from rules expressing only the direct consequences.

We derived the complexity measures for our system, and compared them to measures in the literature for some other important systems.

8.3. Further Research

During this research, we have encountered many problems and questions about problem solving systems, time and event logics, hypothetical reasoning, and the frame problem. Of these problems, we have solved some, and presented these solutions in this thesis. Some of the others have suggested themselves as candidates for hypothetical reasoning based approaches, and this section discusses some of our preliminary ideas for expanding the application of consistency maintenance and hypothetical reasoning to handle these problems.

8.3.1. Simulation with WIF

A simulation for modeling the agents executing the plans generated by our system could also be built using WIF. The simulator code would be very similar to the planner code that describes the domain to the agent. The major difference is that there would presumably be only one possible world from a given set of hypothesis.

The simulator would use the consistency rules in a forward fashion instead of backward in time as the planner does. The hypothesis would be various actions performed by agents, and the results would be the changes caused by the action. The changes would then have to be communicated back to the agent through some means of perception. This would allow the agents to do things and find out whether the results were as expected. Agents could also notice other agents operating in the world, and be surprised by unexpected changes or incorrect information.

The set of facts used by the simulator would have to be separate from those of the agents, so that it could have complete knowledge of the world (although simulating on incomplete knowledge might have interesting applications).

8.3.2. Resources

Resource reasoning is an important ability for agents to possess when making useful plans. Time is probably the most important resource, although there are others, money for instance. There are consumable and renewable resources. Consistency rules are very well suited to expressing the constraints imposed by resource limits and this sort of extension should be rather straightforward once the theoretical aspects are worked out.

8.3.3. Agent World Model

Several important planning problems have to do with independent world models for the agents within the world. Multiple agent problems where the agents interact require some model not only of the world, but of the other agents' models of the world also. This is necessary for any tasks involving cooperation, competition, plan recognition, or communication among agents, to cite a few examples. There is no reason that our approach cannot be combined with existing or future work on representation of world models.

8.3.4. Incomplete/Incorrect Knowledge

Further research needs to be done on the area of incomplete and incorrect knowledge. Incorrect knowledge can be handled when it is discovered already with the consistency maintenance performed by WIF, since any contradictions are noted as soon as an attempt to add them to the fact base is made. The problem with incorrect knowledge is not what to do about it ahead of time: nothing can be done. The difficulty is in finding what other knowledge is affected by a change. WIF is excellent at performing the operation, using the domain rules. The rules allow the relevant facts to be found and checked directly. Other systems have tried such clumsy methods as storage dependency graphs between facts so that facts that become unsupported can be deleted.

Incomplete knowledge is currently handled by creating possible worlds containing the various cases for an unknown predicate. This technique needs some optimization to make it workable on a larger scale, or possibly a new method should be developed to handle it instead, since it can become intractable quickly if applied to a complex domain.

8.3.5. Causality Issues

It might be useful to do a more in-depth analysis of causality. We only really model events causing states. It may make sense to say that events cause other events, or states cause states, or even states causing event, such as an agent intention (a state) causing the intended action (the event). A better model for continuous processes may also be in order, so that we could write more intuitive rules for things like dropping things causes them to hit the floor. We really want to say that unsupported objects accelerate downward, until they hit something. This is not easy to model, although recent work seems to have made progress [Shoham, 1988]. It may be possible to integrate the two approaches.

8.3.6. Hybrid Hierarchical/Consistency-based Planners

Our planner excels at non-linear problems, but it does not solve linear problems in linear time. A hybrid hierarchical/consistency-based planner would be capable of solving goal states by goal ordering until that technique failed, and then switching to consistency-based approaches for resolving difficulties caused by interactions between the goals.

8.3.7. General World Models

It might be possible to model all domains with the same representation language. Humans certainly manage to represent everything they know, and we have to presume that it is all in the same internal language. While such a representation would likely be very inefficient and expensive in terms of computer resources, it could be a very useful tool for theoretical work, and could even become practical if massive parallelism becomes available. It would also have the singular advantage of being applicable to any domain. This could easily work in conjunction with the hierarchical hybrids discussed above different domains require different levels of abstraction but a general system should be able to handle problems at any level.

8.3.8. Incomplete Planning

It certainly must be possible to achieve a faster search time at the expense of completeness. Ultimately, this will probably be necessary to build systems with a 'real' domain, but this type of system should be a good basis for research into the appropriate heuristics for determining relevance, and maybe even for building one. In any case it will be a useful in comparisons to determine in exactly what ways such systems are incomplete and what is gained thereby.

8.3.9. Theoretical Limits on Solutions to the Frame Problem

Now that we have defined the frame problem in computational terms, it is possible to learn a great deal about the limits to solutions of the subproblems. It should be possible to prove what the best complexity measures for the three subproblems as isolated cases. It should also be possible to determine the best solution possible, given particular solutions for the other two. This is very important because the three problems are not independent. These optimal complexity measures will become targets for future systems. Even more importantly, we will know when we have found the 'best' answer and can move on to other questions.

8.3.10. Parallel Implementations for WIF

WIF is ideally suited for a parallel implementation. Each possible world created as it runs is completely independent of the others. Therefore, in a parallel implementation of WIF, no communication would be required between processes, other that at start and completion of the analysis of a possible world. This would avoid the major difficulty with many parallel algorithms, which is the overloading of communication lines between processing nodes.

8.3.11. A Plan Difficulty Metric (PDM)

Once WIF has run, producing the possible worlds that contain the plans for the agent, the agent must decide which plan to follow. This decision should be based on a number of facts, including how many actions are required, the resources required, the difficulty of the individual actions involved, and the expected reliability of the plan, which could be based on incomplete knowledge. It should be possible to create an open-ended numeric scale to represent the difficulty of a plan based on these criteria (and probably others as well). This scale, and the influence of each factor on it would be domain dependent, but useful general techniques may be possible. Ideally, the scale could be designed to be computed such that for the addition of a fact, only one or a few arithmetic operations are required to update the PDM, instead of making it necessary to recompute everything for a small change in the plan world.

8.3.12. Enhancements to Planning with a Parallel WIF

Given a parallel implementation of WIF, as described above, a simple enhancement to the planner would create a much more usable system. As the system stands, it produces all possible worlds, many of which are unreasonable *as plans* when compared to other worlds already generated. The previously discussed PDM could be dynamically calculated by WIF as the planning progresses, simply by a single addition operation for each new event occurrence added to the hypothesis list. Given a single memory location, initialized to the worst case PDM that the planning agent is willing to accept, that is available to all processes, any world that exceeds this value could be immediately thrown out. Any world that terminated with a lower value would set the PDM limit to the new, lower, value. In this manner, the optimal plan can be found very quickly. Another benefit is that infinite loops cannot cause any difficulty, because they involve adding changes, and therefore events, to the world at each step.

8.3.13. Consistency Meta-Rules: Comparing Possible Worlds

WIF, as it stands, cannot compare different possible worlds. No communication between them is possible; neither are any comparisons. If it were possible to compare the states of different possible worlds, then many useful new rules, that we have called consistency meta-rules, would be possible. As an example, one could write a rule that says it is inconsistent create a world that contains a plan more difficult (using the PDM) than a plan already completed. It might also be possible to create rules to make infinite loops impossible if this ability were available. Another possibility is the implementation of heuristic rules to guide the search through possible worlds.

8.4. Conclusion

The problems addressed herein have fallen into three major categories, and all have been approached with hypothetical reasoning in mind as a possible avenue of approach. The three categories are planning, the frame problem, and modeling methods for time and change. We have made significant progress in all three areas.

In the search for a planning algorithm based on HR, we have found both a definition and a solution for the frame problem, which is actually three related problems. Our definition provides a method to compare various attempts to solve the frame problem objectively. It will also be useful for determining the theoretically optimal solutions, and the tradeoffs inherent in the approaches to the three subproblems.

We have defined a system for modeling time, events, and change that is suitable for problem solving systems, and have based our system upon it. We have also identified a complete and sound tense logical system and equivalent modal logic model theory that corresponds to our possible world semantics.

The planner that resulted from this research is capable of handling all types of planning problems identified in the literature up to this point. In addition, it can handle the class of indirect goal problems, which have not been previously explored.

We have also come upon several more possible applications and extensions of this research. These have been discussed in our future research section. We think that the breadth of problems for which we can easily create an initial approach based on consistency maintenance is a sign of its general applicability to the problem solving field, and possibly AI in general.

In the course of this research, we have, on several occasions, developed a method to solve a problem and then noticed afterwards how similar it was to a previously established method. For example, we have noticed similarities to both circumscription (the causality rules), and default reasoning (persistence) in our system. We seem to have combined the beneficial aspects of several techniques, while avoiding their pitfalls.

Appendix A. Planner Code

Following is the actual code for the domain independent portions of the planner. Some of the rules are WIF consistency rules, and some are HC Prolog rules. It should be clear which is which.

```
(assert
;;;;;;;;;;;;;;;;;;;
;
; CREATE_FACTS - bootstrap new basic facts into complete 'closed world'
;                set of facts using WIF and consistency
;                rules for the rest of the system
;
;;;;;;;;;;;;;;;;;;;

; create new facts only if consistent with world as it is
((create_facts *list) if
        (highest_belief *modal)
        (create_facts *list *modal)
        (cut)
        )
; create new facts that are consistent up to a particular modal category
((create_facts *list *modal) if
        (counterfactual *list *modal *results)
        (cut)
        (choose_one *list *results *flattened *)
        (update_facts *flattened)
        )

; create new time based facts if they are consistent with the world
((t_create_facts *list *t) if
        (t_convert_facts *list *t_list *t)
        (create_facts *t_list)
        (cut)
        )
; create new time based facts if they are consistent up to *modal
((t_create_facts *list *t *modal) if
        (t_convert_facts *list *t_list *t)
        (create_facts *t_list *modal)
        (cut)
        )

; convert normal facts to timebased start,stop and state facts
((t_convert_facts nil nil *t))
((t_convert_facts nil *any *t))
((t_convert_facts (*head . *rest) ((start *head *t) . *t_rest) *t) if
        (t_convert_facts *rest *t_rest *t)
        (cut)
        )
```

```
((t_convert_goals nil nil *t))
((t_convert_goals nil *any *t))
((t_convert_goals ((not *head).*rest) ((stop *head *t).*t_rest) *t) if
        (t_convert_goals *rest *t_rest *t)
        (cut)
        )
((t_convert_goals (*head . *rest) ((state *head *t) . *t_rest) *t) if
        (t_convert_goals *rest *t_rest *t)
        (cut)
        )


)

(assert
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; planner setup and WIF call
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

((plan *global *initial *goal *plans) if
        (t_convert_facts *initial *init t0)
        (cut)
        (t_convert_goals *goal *gol t1)
        (cut)
        (= *tmp1 (append ((earlier t0 t1)(depth 0)) *global) )
        (= *tmp2 (append *init *gol) )
        (= *hyp (append *tmp1 *tmp2) )
        (cf_trace_all)
        (cut)
        (counterfactual *hyp 4 *result)
        (cut)
        (set cf_t nil)
        (break "INITIAL PLAN")
        (pprint *result "")
        (flatten *result *dnf)
        (weed-out *dnf *plans)
        (show-plans *plans)
        (break "FINISHED PLANNING")
        )

((weed-out nil nil))                    ; no possible worlds
((weed-out (or . *more) *plans) if      ; multiple possible worlds
        (weed-out2 *more *plans)
        (cut)
        )
((weed-out *single *plans) if           ; one possible world
        (weed-out2 (*single) *plans)
        )

((weed-out2 nil nil))
((weed-out2 (*h . *more) (*plan . *plans)) if
        (counterfactual *h 4 *h)        ; all consequences found
        (cut)
        (print "SUCCESS: ")
        (pprint *h "")
        (get-events *h *events)        ; get the plan
        (t-sort *events *plan)
        (weed-out2 *more *plans)
        (cut)
        )
((weed-out2 (*h . *more) *allplans) if
        (counterfactual *h 4 *result)   ; iterate until *h = *result
        (cut)
        (print "PARTIAL: ")
        (pprint *result "")
        (flatten *result *dnf)
        (weed-out *dnf *plan)
        (weed-out2 *more *plans)
        (= *allplans (append *plan *plans))
        (cut)
        )
((weed-out2 (*h . *more) *plans) if
        (print "FAILURE: ")
        (pprint *h "")
        (weed-out2 *more *plans)        ; throws out inconsistent worlds
        (cut)
        )

((get-events nil nil))
((get-events ((occur *event *t).*more) ((occur *event *t).*rest)) if
        (get-events *more *rest)
        (cut)
        )
((get-events (*h.*more) *rest) if
        (get-events *more *rest)
        )

((t-sort nil nil))
((t-sort ((occur *event *t) . *rest) *sorted) if
        (t-sort *rest *r_sorted)
        (t-insert (occur *event *t) *r_sorted *sorted)
        )

((t-insert *event nil (*event)))
((t-insert (occur *event *t) ((occur *event2 *t2) . *rest)
          ((occur *event *t).((occur *event2 *t2) . *rest)) ) if
        (earlier *t *t2)
        (cut)
        )
((t-insert (occur *event *t) ((occur *event2 *t2) . *rest)
                               ((occur *event2 *t2) . *new) ) if
        (t-insert (occur *event *t) *rest *new)
        (cut)
        )

((show-plans nil))
((show-plans (*h . *t)) if
        (print "PLAN: " *h)
        (show-plans *t)
        (cut)
        )
)


(assert

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; persistence rules
;    these rules delay the derivation of things until asked for
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

((fact (not (start *fact *t)) *mc) if
      (look_for (start *fact *t) *mc)
      (cut)
      (fail))
((fact (not (start *fact *t)) *mc) if
        (look_for (state *fact *t) *mc))
((fact (not (start *fact *t)) *mc) if
      (cut))
((fact (not (stop *fact *t)) *mc) if
      (look_for (stop *fact *t) *mc)
      (cut)
      (fail))
((fact (not (stop *fact *t)) *mc) if
        (look_for (not (state *fact *t)) *mc))
((fact (not (stop *fact *t)) *mc) if
      (cut))

((fact (state *fact t0) *mc) if
        (look_for (start *fact t0) *mc))
((fact (state *fact *t) *mc) if
        (!= *t t0)
        (look_for (stop *fact *t) *mc)
        )
((fact (state *fact *t) *mc) if
        (get_modal_cat (state.*any) *mc)
        (fact (persist (state *fact *t) *tprev) 0)
      )

((fact (persist (state *fact *t) *tprev) 0) if ; state persists until *t
        (look_for (earlier *tprev *t) *mc1)
        (look_for (start *fact *tprev) *mc2)   ; it was true at *tprev
        (not_stopped *fact *t *tprev)          ; and hasn't changed
        (cut)
        )

; STOPPED if there is a STOP for *fact between *tprev and *t
((not_stopped *fact *t *tprev) if
        (look_for (during *tprev *tt *t) *mc1)
        (look_for (stop *fact *tt) *mc2)
        (cut)
        (fail))
((not_stopped *fact *t *tprev) if
        (cut))

; persistence for false facts - including persistence for unspecified
;                                 facts that WIF assumes false
((fact (not (state *fact *t)) *mc) if
        (look_for (state *fact *t) *mc)
        (cut)
        (fail))
((fact (not (state *fact t0)) *mc) if
        (look_for (stop *fact t0) *mc))
((fact (not (state *fact *t)) *mc) if
        (get_modal_cat (not (state.*any)) *mc)
        (fact (persist (not (state *fact *t)) *tprev) 0))

; persistence for not state

((fact (persist (not (state *fact *t)) *tprev) 0) if ;not state persists
        (look_for (earlier *tprev *t) *mc1)          ;if it was false
        (look_for (stop *fact *tprev) *mc2)
        (not_started *fact *t *tprev)                ;and hasn't changed
        (get_modal_cat (state.*any) *mc)
        (cut)
        )

; closed world assumption for persistence of not state at t0

((fact (persist (not (state *fact *t)) t0) 0) if
        (fact (persist (state *fact *t) *prev) 0)
        (cut)
        (fail))
((fact (persist (not (state *fact *t)) t0) 0) if
        (cut))

; STARTED if there is a START for *fact between *tprev and *t
((not_started *fact *t *tprev) if
        (look_for (start *fact *t) *mc)
        (cut)
        (fail))
((not_started *fact *t *tprev) if
        (look_for (during *tprev *tt *t) *mc1)
        (look_for (start *fact *tt) *mc2)
        (cut)
        (fail))
((not_started *fact *t *tprev) if
        (cut))
;; get_modal_cat facts
((get_modal_cat (not (isa.*any))  5))
((get_modal_cat (isa.*any)  5))
((get_modal_cat (not (attribute.*any))  5))
((get_modal_cat (attribute.*any)  5))

((get_modal_cat (not (check.*any)) 0))
((get_modal_cat (check.*any) 0))
```

```
((get_modal_cat (not (get_start.*any)) 5))
((get_modal_cat (get_start.*any) 5))
((get_modal_cat (not (get_stop.*any)) 5))
((get_modal_cat (get_stop.*any) 5))
((get_modal_cat (not (get_thing.*any)) 5))
((get_modal_cat (get_thing.*any) 5))
((get_modal_cat (not (during.*any)) 5))
((get_modal_cat (during.*any) 5))
((get_modal_cat (not (earlier.*any)) 5))
((get_modal_cat (earlier.*any) 5))
((get_modal_cat (not (maybe_earlier.*any)) 5))
((get_modal_cat (maybe_earlier.*any) 5))
((get_modal_cat (not (depth.*any)) 5))
((get_modal_cat (depth.*any) 5))

((get_modal_cat (not (persist.*any)) 0))
((get_modal_cat (persist.*any) 0))

((get_modal_cat (+.*any) 0))
((get_modal_cat (<.*any) 0))
((get_modal_cat (=.*any) 0))
((get_modal_cat (<>.*any) 0))
((get_modal_cat (not (+.*any)) 0))
((get_modal_cat (not (<.*any)) 0))
((get_modal_cat (not (=.*any)) 0))
((get_modal_cat (not (<>.*any)) 0))

;; math facts
((fact (< *a *b) 0) if
        (boundp *a)
        (boundp *b)
        (< *a *b)
        (cut)
        )
((fact (+ *a *b *c) 0) if
        (boundp *a)
        (boundp *b)
        (:= *c (+ *a *b))
        (cut)
        )
((fact (+ *a *b *c) 0) if
        (boundp *a)
        (boundp *c)
        (:= *b (+ *a *c))
        (cut)
        )
((fact (+ *a *b *c) 0) if
        (boundp *b)
        (boundp *c)
        (:= *a (+ *c *b))
        (cut)
        )

((fact (get_thing *thing *check) *mc) if
        (unboundp *thing)
        (look_for (isa thing *thing) *mc)
        (== *check ok))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; return appropriate times between start and end
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

((fact (get_start *start *start *end *start *end *depth *check) 0) if
        (!= *start t0))
((fact (get_start *start *t *end *start *end *depth *check) 0) if
        (unboundp *t)
        (unboundp *depth)
        (look_for (during *start *t *end) *mc)
        (!= *t *end)
        (greatest_depth *depth)
        (== *check ok))      ; instantiation will only be able to bind
                             ; *check when these rules do the binding for
                             ; the other time variables
((fact (get_start *start *t *end *a *b *new_depth *check) 0) if
        (get_during *start *t *end *a *b *new_depth *check))

((fact (get_stop *start *t *end *start *end *depth *check) 0) if
        (unboundp *t)
        (unboundp *depth)
        (look_for (during *start *t *end) *mc)
        (greatest_depth *depth)
        (== *check ok))
((fact (get_stop *start *t *end *a *b *new_depth *check) 0) if
        (get_during *start *t *end *a *b *new_depth *check))
((get_during *start *t *end *a *b *new_depth *check) if
        (unboundp *t)
        (unboundp *a)
        (unboundp *b)
        (unboundp *new_depth)
        (get_new *start *t *end *a *b)
        (greatest_depth *depth)
        (:= *new_depth (+ 1 *depth))
        (== *check ok))

((unboundp *b) if
        (boundp *b)
        (cut)
        (fail))
((unboundp *b) if
        (cut))

((greatest_depth *depth) if
        (cntfact (depth *depth))
        (none_greater *depth))
((none_greater *depth) if
        (cntfact (depth *x))
        (> *x *depth)
        (cut)
        (fail))
((none_greater *depth) if
        (cut))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; return a new time in each interval between start and end
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; if only one interval
((get_new *start *t *end *start *end) if
        (= *t (gensym))
        (look_for (earlier *start *end) *mc)
        (not_both (earlier *start *x)
                  (earlier *x *end) ))
((get_new *start *t *end *start *end) if
        (boundp *t)
        (look_for (earlier *start *end) *mc)
        (not_both (earlier *start *x)
                  (earlier *x *end) ))

; first of two or more intervals
((get_new *start *t *end *start *first) if
        (boundp *t)
        (look_for (earlier *start *end) *mc1)
        (look_for (earlier *start *first) *mc2)
        (look_for (earlier *first *end) *mc3)
        (not_both (earlier *start *x)
                  (earlier *x *first) ))
((get_new *start *t *end *start *first) if
        (= *t (gensym))
        (look_for (earlier *start *end) *mc1)
        (look_for (earlier *start *first) *mc2)
        (look_for (earlier *first *end) *mc3)
        (not_both (earlier *start *x)
                  (earlier *x *first) ))

; other intervals
((get_new *start *t *end *a *b) if
        (boundp *t)
        (look_for (earlier *start *first) *mc1)
        (look_for (earlier *first *end) *mc2)
        (not_both (earlier *start *x)
                  (earlier *x *first))
        (get_new *first *t *end *a *b)
        )
((get_new *start *t *end *a *b) if
        (= *t (gensym))
        (look_for (earlier *start *first) *mc1)
        (look_for (earlier *first *end) *mc2)
        (not_both (earlier *start *x)
                  (earlier *x *first))
        (get_new *first *t *end *a *b)
        )

((not_both *fact1 *fact2) if
        (look_for *fact1 *mc1)
        (look_for *fact2 *mc2)
        (cut)
        (fail))
((not_both *fact1 *fact2) if
        (cut))

) ; end of assert


(mk_rules

; ISA rules
((earlier *t1 *t2)
        ->> (isa time *t1)
            (isa time *t2))
((attribute *attr *val1 *thing)
 (attribute *attr *val2 *thing)
 (<> *val1 *val2)
        ->> (inconsistent))

; EARLIER rules
((earlier *t t0)                ; first time
        ->> (inconsistent))
((earlier t1 *t)                ; last time
        ->> (inconsistent))
((earlier *t1 *t2)              ; antisymetric & irreflexive
 (earlier *t2 *t1)
        ->> (inconsistent))
((earlier *t1 *t2)              ; transitive
 (earlier *t2 *t3)
        ->> (earlier *t1 *t3))
((isa time *t1)                 ; total order
 (isa time *t2)
 (<> *t1 *t2)
 (not (earlier *t1 *t2))
 (not (earlier *t2 *t1))
        ->> (inconsistent))

; DURING rules
((earlier *t1 *t2)
 (earlier *t2 *t3)
        ->> (during *t1 *t2 *t3))
((earlier *t1 *t2)
        ->> (during *t1 *t2 *t2))
((during *t1 *t2 *t3)
        ->> (earlier *t1 *t2))
((during *t1 *t2 *t3)
 (<> *t2 *t3)
        ->> (earlier *t2 *t3))

; every change has to really be a change
((start *fact *t)
 (stop *fact *t)
        ->> (inconsistent))
```

```
((start *fact *t)
 (<> *t t0)
       ->> (not (state *fact *t)))
((stop *fact *t)
 (<> *t t0)
       ->> (state *fact *t))


; if state is true and its time ≠ t0 and it was false then it started
((state *fact *t)
 (<> t0 *t)
 (persist (not (state *fact *t)) *tprev)
       ->> (get_start *tprev *tstart *t *ta *tb *depth *check)
           (depth *depth)
           (maybe_earlier *ta *tstart)
           (earlier *tstart *tb)
           (start *fact *tstart))
((not (state *fact *t))
 (<> t0 *t)
 (persist (state *fact *t) *tprev)
       ->> (get_stop *tprev *tstop *t *ta *tb *depth *check)
           (depth *depth)
           (maybe_earlier *tstop *tb)
           (earlier *ta *tstop)
           (stop *fact *tstop))
((maybe_earlier *ta *tb)
 (<> *ta *tb)
       ->> (earlier *ta *tb))

) ; end of mk_rules


(mk_facts -1 (not (inconsistent)))
```

Appendix B. Blocks World Domain Rules

These are the domain rules for the Blocks World domain. They are WIF consistency rules that describe blocks world.

```
(assert

((get_modal_cat (not (state *fact *t)) 7))
((get_modal_cat (state *fact *t) 7))
((get_modal_cat (not (start *fact *t)) 7))
((get_modal_cat (start *fact *t) 7))
((get_modal_cat (not (stop *fact *t)) 7))
((get_modal_cat (stop  *fact *t) 7))
((get_modal_cat (not (occur *fact *t)) 7))
((get_modal_cat (occur *fact *t) 7))

) ; end of assert

(mk_rules

((isa block *thing)
        ->> (isa thing *thing)
            (attribute size small *thing))
((isa table *thing)
        ->> (isa thing *thing)
            (attribute size large *thing))

; every change of any predicate has a cause: ON
((start (on *a *b) *t)
 (<> *t t0)
        ->> (occur (move *a *b) *t))
((stop (on *a *b) *t)
 (<> *t t0)
        ->> (get_thing *c *check)
            (occur (move *a *c) *t))

; no two things in same place
((start (on *a *c) *t)
 (state (on *b *c) *t)
 (<> *a *b)
 (not (attribute size large *c))
      ->> (stop (on *b *c) *t))

; nothing in two places
((start (on *a *b) *t)
 (state (on *a *c) *t)
 (<> *b *c)
      ->> (stop (on *a *c) *t))
```

```
; no two events at the same time since we have a single agent
((occur *event1 *t)
 (occur *event2 *t)
 (<> *event1 *event2)
        ->> (inconsistent))

; restrictions on movement on blocks
((occur (move *thing *place) *t)
        ->> (<> *thing *place))
((occur (move *thing *place) *t)
        ->> (attribute size small *thing))
((occur (move *thing *place) *t)
        ->> (start (on *thing *place) *t))
((occur (move *thing *place) *t)
 (persist (state (on *anything *thing) *t) *tprev)
        ->> (not (state (on *anything *thing) *t)))

; over rules
; irreflexive (and antisymmetric with help from transitivity)
((start (over *a *a) *t)
        ->> (inconsistent))

; if *a starts being over *b at a time point before which it already
;     persists, then inconsistent
((start (over *a *b) *t)
 (<> t0 *t)
 (persist (state (over *a *b) *t) *tprev)
      ->> (inconsistent))

; transitive (unless already over)
((start (over *a *b) *t)
 (state (on *b *c) *t)
 (persist (not (state (over *a *c) *t)) *tprev)
        ->> (start (over *a *c) *t))
((start (over *b *c) *t)
 (state (on *a *b) *t)
 (persist (not (state (over *a *c) *t)) *tprev)
        ->> (start (over *a *c) *t))

; on -> over (unless already over)
((start (on *a *b) *t)
 (persist (not (state (over *a *b) *t)) *tprev)
        ->> (start (over *a *b) *t))

; 'causal'
((start (over *a *b) *t)
 (<> t0 *t)
 (not (start (on *a *b) *t))
       ->> (get_thing *c *check)
           (start (on *a *c) *t)
           (state (over *c *b) *t))
((stop (over *a *b) *t)
 (<> t0 *t)
 (not (stop (on *a *b) *t))
```

```
        ->> (get_thing *c *check)
            (start (on *a *c) *t)
            (not (state (over *c *b) *t)))

) ; end of mk_rules
```

Appendix C. Test Plans and Results

The results of each of our blocks world planner tests follows. Each test is first defined, and then it is followed by the actual output from the planner using the rules in Appendices A and B. The problem is stated in the call to WIF in the same format as the examples. The results are presented as inference trees. Each hypothesis is the root of a tree. The direct consequences of each fact follow an arrow, and are indented beneath the fact at the same indention level as the arrow. Each Poblem starts on a new page. They start off very simple and become more challenging.

```
((test1a) if
        (plan ((isa block a)(isa block b)(isa table t)(not (depth 2)))
        ((on a b)(on b t))
        ((on a t))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 2))
 ((start (on a b) t0)
   =>
   (start (over a b) t0)
    =>
    (start (over a t) t0))
((start (on b t) t0)
  =>
  (start (over b t) t0))
((state (on a t) t1)
  =>
  (get_start t0 G101 t1 t0 t1 1 ok)
  (depth 1)
  ((maybe_earlier t0 G101)
    =>
    (earlier t0 G101)
     =>
     (isa time G101)
     (during t0 G101 t1)
     (during t0 G101 G101))
  ((earlier G101 t1)
    =>
    (during G101 t1 t1))
  ((start (on a t) G101)
    =>
    (stop (on a b) G101)
     =>
     (get_thing t ok)
     (occur (move a t) G101))))
```

```
((test1b) if
        (plan ((isa block a)(isa block b)(isa table t)(not (depth 2)))
        ((on a t)(on b t))
        ((on a b))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 2))
 ((start (on a t) t0)
   =>
   (start (over a t) t0))
 ((start (on b t) t0)
   =>
   (start (over b t) t0))
 ((state (on a b) t1)
   =>
   (get_start t0 G94 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G94)
     =>
     (earlier t0 G94)
      =>
      (isa time G94)
      (during t0 G94 t1)
      (during t0 G94 G94))
   ((earlier G94 t1)
     =>
     (during G94 t1 t1))
   ((start (on a b) G94)
     =>
     ((stop (on a t) G94)
       =>
       (get_thing b ok)
       (occur (move a b) G94))
     (start (over a b) G94))))
```

```
((test2) if
       (plan ((isa block a)(isa block b)(isa table t)(not (depth 3)))
        ((on a b)(on b t))
        ((on b a))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 3))
 ((start (on a b) t0)
   =>
   (start (over a b) t0)
    =>
    (start (over a t) t0))
 ((start (on b t) t0)
   =>
   (start (over b t) t0))
 ((state (on b a) t1)
   =>
   (get_start t0 G311 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G311)
     =>
     (earlier t0 G311)
      =>
      (isa time G311)
      (during t0 G311 t1)
      (during t0 G311 G311))
   ((earlier G311 t1)
     =>
     (during G311 t1 t1))
   ((start (on b a) G311)
     =>
     ((stop (on b t) G311)
       =>
       (get_thing a ok)
       ((occur (move b a) G311)
         =>
         (not (state (on a b) G311))
      =>
      (get_stop t0 G593 G311 t0 G311 2 ok)
      (depth 2)
      ((maybe_earlier G593 G311)
        =>
        (earlier G593 G311)
         =>
         (isa time G593)
         ((earlier G593 t1)
           =>
           (during t0 G593 t1)
           (during G593 t1 t1))
         (during G593 G311 t1)
         (during t0 G593 G311)
         (during G593 G311 G311))
      ((earlier t0 G593)
        =>
        (during t0 G593 G593))
      ((stop (on a b) G593)
        =>
        (get_thing t ok)
        ((occur (move a t) G593)
          =>
          (start (on a t) G593)))))
(start (over b a) G311))))
```

```
((test2a) if
        (plan ((isa block a)(isa block b)(isa block c)(isa table t)
                        (not (depth 3)))
        ((on a b)(on b c)(on c t))
        ((on a t)(on b t))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa block c)
   =>
   (isa thing c)
   (attribute size small c))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 3))
 ((start (on a b) t0)
   =>
   (start (over a b) t0)
    =>
    (start (over a c) t0)
     =>
     (start (over a t) t0))
 ((start (on b c) t0)
   =>
   (start (over b c) t0)
    =>
    (start (over b t) t0))
 ((start (on c t) t0)
   =>
   (start (over c t) t0))
 ((state (on a t) t1)
   =>
   (get_start t0 G140 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G140)
     =>
     (earlier t0 G140)
      =>
      (isa time G140)
      (during t0 G140 t1)
```

```
        (during t0 G140 G140))
     ((earlier G140 t1)
       =>
       (during G140 t1 t1))
     ((start (on a t) G140)
       =>
       ((stop (on a b) G140)
         =>
         (get_thing t ok)
         (occur (move a t) G140)))
  ((state (on b t) t1)
    =>
     (get_start t0 G643 t1 G140 t1 2 ok)
     (depth 2)
     ((maybe_earlier G140 G643)
       =>
       (earlier G140 G643)
        =>
        (isa time G643)
        ((earlier t0 G643)
          =>
          (during t0 G643 t1)
          (during t0 G643 G643))
        (during G140 G643 t1)
        (during t0 G140 G643)
        (during G140 G643 G643))
     ((earlier G643 t1)
       =>
       (during G643 t1 t1))
     ((start (on b t) G643)
       =>
       (stop (on b c) G643)
        =>
        (occur (move b t) G643)
         =>
         (not (state (on a b) G643))
          =>
          (get_stop t0 G140 G643 t0 G643 2 ok)
          (stop (on a b) G140))))
```

```
((test2b) if
       (plan ((isa block a)(isa block b)(isa block c)(isa table t)
                           (not (depth 3)))
       ((on a b)(on b c)(on c t))
       ((on b t)(on a t))
       *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa block c)
   =>
   (isa thing c)
   (attribute size small c))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 3))
 ((start (on a b) t0)
   =>
   (start (over a b) t0)
    =>
    (start (over a c) t0)
     =>
     (start (over a t) t0))
 ((start (on b c) t0)
   =>
   (start (over b c) t0)
    =>
    (start (over b t) t0))
 ((start (on c t) t0)
   =>
   (start (over c t) t0))
 ((state (on b t) t1)
   =>
   (get_start t0 G140 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G140)
     =>
     (earlier t0 G140)
      =>
      (isa time G140)
      (during t0 G140 t1)
      (during t0 G140 G140))
   ((earlier G140 t1)
     =>
     (during G140 t1 t1))
   ((start (on b t) G140)
     =>
     ((occur (move b t) G140)
       =>
       (not (state (on a b) G140))
        =>
        (get_stop t0 G218 G140 t0 G140 2 ok)
        (depth 2)
        ((maybe_earlier G218 G140)
          =>
          (earlier G218 G140)
           =>
           (isa time G218)
           ((earlier G218 t1)
             =>
             (during t0 G218 t1)
             (during G218 t1 t1))
           (during G218 G140 t1)
           (during t0 G218 G140)
           (during G218 G140 G140))
        ((earlier t0 G218)
          =>
          (during t0 G218 G218))
        ((stop (on a b) G218)
          =>
          ((get_thing t ok)
           ((occur (move a t) G218)
             =>
             (start (on a t) G218)))
    (stop (on b c) G140))))
 ((state (on a t) t1)
((testCAB) if
        (plan ((isa block a)(isa block b)(isa block c)(isa table t)
                        (not (depth 4)))
        ((on c a)(on a t)(on b t))
        ((on a b)(on b c))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa block c)
   =>
   (isa thing c)
   (attribute size small c))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 4))
 ((start (on c a) t0)
   =>
   (start (over c a) t0)
    =>
    (start (over c t) t0))
 ((start (on a t) t0)
   =>
   (start (over a t) t0))
 ((start (on b t) t0)
   =>
   (start (over b t) t0))
 ((state (on a b) t1)
   =>
   (get_start t0 G1009 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G1009)
     =>
     (earlier t0 G1009)
      =>
      (isa time G1009)
      (during t0 G1009 t1)
      (during t0 G1009 G1009))
   ((earlier G1009 t1)
     =>
     (during G1009 t1 t1))
```

```
  ((start (on a b) G1009)
    =>
    ((stop (on a t) G1009)
      =>
      (get_thing b ok)
      ((occur (move a b) G1009)
        =>
        (not (state (on c a) G1009))
         =>
         (get_stop t0 G1803 G1009 t0 G1009 2 ok)
         (depth 2)
         ((maybe_earlier G1803 G1009)
           =>
           (earlier G1803 G1009)
            =>
            (isa time G1803)
            ((earlier G1803 t1)
              =>
              (during t0 G1803 t1)
              (during G1803 t1 t1))
            (during G1803 G1009 t1)
            (during t0 G1803 G1009)
            (during G1803 G1009 G1009))
         ((earlier t0 G1803)
           =>
           (during t0 G1803 G1803))
         ((stop (on c a) G1803)
           =>
           ((get_thing t ok)
            ((occur (move c t) G1803)
              =>
              (start (on c t) G1803)))))))
    (start (over a b) G1009)))
((state (on b c) t1)
  =>
  (get_start t0 G1989 t1 t0 t1 3 ok)
  (maybe_earlier t0 G1989)
  ((start (on b c) G1989)
    =>
    ((stop (on b t) G1989)
      =>
      (get_thing c ok)
      (occur (move b c) G1989))
    ((start (over b c) G1989)
      =>
      (start (over b a) G1989)))))
```

```
((testIndGoal) if
        (plan ((isa block a)(isa block b)(isa block c)(isa table t)
                        (not (depth 2)))
        ((on a b)(on b t)(on c t))
        ((over c b))
        *plan))

(((earlier t0 t1)
   =>
   ((isa time t0)
    (isa time t1))
   (during t0 t1 t1))
 (depth 0)
 ((isa block a)
   =>
   (isa thing a)
   (attribute size small a))
 ((isa block b)
   =>
   (isa thing b)
   (attribute size small b))
 ((isa block c)
   =>
   (isa thing c)
   (attribute size small c))
 ((isa table t)
   =>
   (isa thing t)
   (attribute size large t))
 (not (depth 2))
 ((start (on a b) t0)
   =>
   (start (over a b) t0)
    =>
    (start (over a t) t0))
 ((start (on b t) t0)
   =>
   (start (over b t) t0))
 ((start (on c t) t0)
   =>
   (start (over c t) t0))
 ((state (over c b) t1)
   =>
   (get_start t0 G124 t1 t0 t1 1 ok)
   (depth 1)
   ((maybe_earlier t0 G124)
     =>
     (earlier t0 G124)
      =>
      (isa time G124)
      (during t0 G124 t1)
      (during t0 G124 G124))
   ((earlier G124 t1)
     =>
     (during G124 t1 t1))
((start (over c b) G124)
  =>
  (get_thing a ok)
  ((start (on c a) G124)
    =>
    (occur (move c a) G124)
    (stop (on c t) G124)
    (start (over c a) G124)))))
```

Bibliography

Allen, James F. "An Interval-Based Representation of Temporal Knowledge". *Proceedings, Seventh International Joint Conference on AI*, Vancouver, Canada, pp. 221-226, 1981.

Allen, James F. and Hayes, Patrick J. *Moments and Points in an Interval Based Temporal Logic*. University of Rochester TR 180, December 1987.

Barwise, Jon and Perry, John. *Situations and Attitudes*. Cambridge, Massachusetts: MIT Press, 1983.

Benthem, J. F. A. K. van. *The Logic of Time*. Dordrecht, Holland: D. Reidel Publishing Company, 1983.

Bratman, Michael E. *Intention, Plans, and Practical Reason*. Cambridge, Massachusets: Harvard University Press, 1987.

Chapman, David. *Planning for Conjunctive Goals*. MIT AI Laboratory Technical Report 802, 1985.

Cohen, P. *Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach*. London, England: Pittman Publishing Ltd., 1985.

Dreben, Burton and Goldfarb, Warren D. *The Decision Problem*. Reading, Massachusets: Addison-Wesley Publishing Company, Inc, 1979.

Fikes, Richard E. and Nilsson, Nils J. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving." *Artificial Intelligence* 2 (1971), 189-208.

Galton, Anthony. *The Logic of Aspect*. Oxford, England: Clarendon Press, 1984.

Genesereth, Michael R. and Nilsson, Nils J. *Logical Foundations of Artificial Intelligence*. Los Altos, California: Morgan Kaufman, 1987.

Ginsberg, Matthew L. and Smith, David E. "Reasoning About Action I: A Possible Worlds Approach". *Artificial Intelligence* 35 (1988), 165-195. Amsterdam, Holland: North-Holland, 1988a.

Ginsberg, Matthew L. and Smith, David E. "Reasoning About Action II: The Qualification Problem". *Artificial Intelligence* 35 (1988), 311-342. Amsterdam, Holland: North-Holland, 1988b.

Goldblatt, Roger. *Logics of Time and Computation.* Stanford, California: Center for the Study of Language and Information, 1987.

Graham, Robert. "A Simulation for Multi-agent Robot Problem Solving" Baccalaureate Honors Thesis, Department of Computer Science, Virginia Tech, 1986.

Hanks, S. and McDermott, D. "Nonmonotonic Logic and Temporal Projection". *Artificial Intelligence* 33 (1987), 379-412. Amsterdam, Holland: North-Holland, 1987.

Hayes, Patrick J. "The Frame Problem and Related Problems in Artificial Intelligence", *Artificial and Human Thinking*, Alick Elithorne and David Jones, editors. pp. 45-59. San Fransisco, California: Jossey-Bass/Elsevier, 1973.

Hintikka, Jaakko. *Model Minimization - An Alternative to Circumscription.* Dordrecht, Holland: D. Reidel Publishing Company, 1988.

Hughes, G. E., and Cresswell, M. J. *A Companion to Modal Logic.* New York, New York: Methuen Press, 1984.

Humberstone, I. L. "An Interval Semantics for Tense Logic: Some Remarks" *Journal of Philosophical Logic 8 (1979).* D. Reidel Publishing Company, 1979.

Hunter, Geoffrey. *Metalogic, An Introduction to the Metatheory of Standard First Order Logic.* Berkeley, California: University of California Press, 1973.

Joslin, David E. and Roach, John W. *An Analysis of Conjunctive Goal Planning.* TR-86-34, Department of Computer Science, Virginia Tech, 1986.

Kowalski, Robert and Sergot, Marek. *A Logic-Based Calculus of Events.* Imperial College, London, 1985.

Kvart, Igal. *A Theory of Counterfactuals.* Indianapolis, Indiana: Hackett Publishing Company, 1981.

Lewis, David. *Counterfactuals.* Cambridge, Massachusetts: Harvard University Press, 1973.

McArthur, R. P. *Tense Logic*, Synthese Library, vol. iii. Dordrecht, Holland: D. Reidel Publishing Co., 1976.

McCarthy, John. "Programs with Common Sense", *Semantic Information Processing*, edited by Marvin Minsky, 403-418, Cambridge, Massachusets: MIT Press, 1968.

McCarthy, John, and Hayes, Patrick. "Some Philosophical Problems from the Standpoint of Artificial Intelligence". Machine Intelligence 4, 463-502. New York, New York: Halstead Press, 1969.

McCarthy, John. "Circumscription - A Form of Nonmonotonic Reasoning". *Artificial Intelligence* 13 (1980) 27-39. Amsterdam, Holland: North-Holland, 1980.

McDermott, Drew. "A Temporal Logic for Reasoning about Plans and Actions". *Cognitive Science* 6, pp. 101-155. Norwood, New Jersey: Ablex, 1982.

Morris, Paul. "A Truth-Maintenance Based Approach to the Frame Problem". *The Frame Problem in Artificial Intelligence*, edited by Frank M. Brown, 297-307. Los Altos, California: Morgan Kaufman Publishers, 1987.

Pelavin, Richard N. and Allen, James F. "A Model for Concurrent Actions Having Temporal Extent." *Proceedings, AAAI 1987*, 246-250, 1987.

Prior, Arthur N. *Past, Present, and Future*. Oxford, England: Clarendon Press, 1967a.

Prior, Arthur N. *Time and Tense*. Oxford, England: Clarendon Press, 1967b.

Raphael, B. "The Frame Problem in Problem-Solving Systems". *Heuristic Programming for AI*, edited by Nicholas Findler, 159-169. Amsterdam, Holland: North-Holland, 1973.

Reiter, R. "On Closed World Databases". *Logic and Databases*, edited by H. Gallaire and J. Minker, 55-76. New York, New York: Plenum Press: 1978.

Reiter, R. "A Logic for Default Reasoning". *Artificial Intelligence* 13 (1980) 81-132. Amsterdam, Holland: North-Holland, 1980.

Rescher, Nicholas. *Hypothetical Reasoning*. Amsterdam, Holland: North-Holland, 1964.

Rescher, Nicholas and Urquhart, Alasdair. *Temporal Logic*. Berlin, West Germany: Springer-Verlag, 1971.

Roach, John W. "The JANITOR Robot". Internal Report, University of Texas at Austin, 1972.

Roach, John W. "A Plan Executing Robot Simulation". Internal Report, University of Texas at Austin, 1975.

Roach, John W. "Coordinating Multiple Parallel Processes", *Proceedings, 25th IEEE Conference on Decision and Control.*, Athens, Greece: 1986.

Roach, John W. "Hypothetical Reasoning Applied to AI Problem Solving". Funded CIT research proposal, 1987.

Roach, John W., Eichelman II, Fred, and Whitehead, J. Douglass H. "A Coherence Logic for Counterfactual Reasoning" Department of Computer Science, Virginia Tech, 1985.

Sacerdoti, Earl D. "The Nonlinear Nature of Plans". *IJCAI* 4, 206-214, 1975.

Sacerdoti, Earl D. *A Structure for Plans and Behavior.* San Fransisco, California: American Elsevier Publishing Company, 1977.

Schwind, Camilla B. "Action Theory and the Frame Problem". *The Frame Problem in Artificial Intelligence,* edited by Frank M. Brown, 121-134. Los Altos, California: Morgan Kaufman Publishers, 1987.

Shoham, Yoav. *Reasoning About Change.* Cambridge, Massachusets: MIT Press, 1988.

Siklossy, L. and Dreussi, J. *A Hierarchy-Driven Robot Planner which Generates Its Own Procedures.* University of Texas at Austin Technical Report TR-6, 1973. Reprinted in *IJCAI* 3, Stanford: 1973

Stalnaker, Robert C. "A Theory of Counterfactuals". *Studies in Logical Theory* edited by Nicholas Rescher. Oxford: Blackwell Press, 1968.

Tate, Austin. "Generating Project Networks". *IJCAI* 5, 888-893, 1977.

Turner, Raymond, "Temporal Logic in Artificial Intelligence". *Logics for AI.* Chichester, England: Ellis-Horwood, 1984.

Whitehead, J. Douglass H. *Fault Diagnosis Based on Causal Reasoning.* Master's Thesis, Department of Computer Science, Virginia Tech. 1988.

Whitehead, J. Douglass H. and Roach, John. "Expert Systems without the Expert: Fault Diagnosis Based on Causal Reasoning". *Texas Instruments Technical Journal,* Winter 1987, pp. 19-29.

The vita has been removed from
the scanned document