Knowledge Representation and Reasoning for Mixed-Initiative Planning

by

George Montague Ferguson

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Professor James F. Allen

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

1995

Acknowledgments

I would like the acknowledge first and foremost the support of my supervisor, James Allen, over my five or so years at Rochester. The breadth of his perspective made the work interesting, and his encouragement kept me going when I might otherwise have given up.

I would also like to acknowledge the other members of my committee: Len Schubert, who was always to ready to discuss the important issues and who taught me to "take language seriously," Henry Kyburg, who always had something interesting to say about the newcomers in the non-mon community crowding his turf, and Ron Loui, whose writings on defeasible reasoning inspired much of my work on arguing about plans and whose writing style I cannot ever hope to imitate. I would also like to acknowledge the influence of my M.S. supervisor at Alberta, Randy Goebel, who got me started checking for tigers under tables. Of course, none of them is responsible for any remaining errors or omissions.

The CS Department at Rochester is a unique place for many reasons, but first and foremost because of the people. I would like to thank Chris Brown for the films, the good times, and the shop talk; Lambert Wixson, for being my 400 TA and a good friend ever since; Marc Light, Polly Pook, and Mark Crovella, my co-conspirators from 1989 (from Backstreets to BU—what a trip); Graham Katz for helping me pace myself; Tom Leblanc for the hockey; Nat Martin for being able to talk both shop and hops, and Sherri, Hannah, and Gracie for coming to happy hour and letting me show off my bellybutton; and the original TRAINS group: Massimo Poesio, Dave Traum, Chung Hee Hwang, Peter Heeman, Brad Miller, Marc, Nat, Len, and James. Finally, the people who make the department run are the lab staff: Liud Bukys, Tim Becker, Jim Roche, and Ray Frank, and the secretarial staff: Jill Forster, Peggy Frantz, Peg Meeker, and Pat Marshall. Thanks all.

This material is based upon work supported by the National Science Foundation under Grant number IRI-9003841. The Government has certain rights in this material. This work is also supported by ONR/ARPA research grant no. N00014-92-J-1512 and Air Force – Rome Air Development Center research contract no. F30602-91-C-0010.

Finally, I would like to thank my parents for their support through seven years of grad school. And last but far from least, I would like to thank Pia for her love and support over the years.

Abstract

This dissertation describes the formal foundations and implementation of a commonsense, mixed-initiative plan reasoning system. By "plan reasoning" I mean the complete range of cognitive tasks that people perform with plans including, for example, plan construction (planning), plan recognition, plan evaluation and comparison, and plan repair (replanning), among other things. "Mixed-initiative" means that several participants can each make contributions to the plan under development through some form of communication. "Commonsense" means that the system represents plans and their constituents at a level that is "natural" to us in the sense that they can be described and discussed in language. In addition, the reasoning that the system performs includes those conclusions that we would take to be sanctioned by common sense, including especially those conclusions that are defeasible given additional knowledge or time spent reasoning.

The main theses of this dissertation are the following:

1. Any representation of plans sufficient for commonsense plan reasoning must be based on an expressive and natural representation of such underlying phenomena as time, properties, events, and actions.

2. For mixed-initiative planning, plans should be viewed as arguments that a certain course of action under certain conditions will achieve certain goals.

These theses are defended by presenting, first, a representation of events and actions based on interval temporal logic and, second, a representation of plans as arguments in a formal system of defeasible reasoning that explicitly constructs arguments. These two aspects of commonsense plan reasoning are combined and implemented in the TRAINS domain plan reasoner, which is also described in detail.

The emphasis in this dissertation is on breadth, taking as its data human communicative and plan reasoning abilities and developing formalisms that characterize these abilities and systems that approximate them. I therefore draw on literature from a broad range of disciplines in the development of these ideas, including: philosophy of language, linguistics and AI work on knowledge representation for the representation of events and actions, philosophical logic and AI work on nonmonotonic reasoning for representing defeasible knowledge and reasoning about it, and, of course, AI work on planning and planning recognition itself.

Table of Contents

List of Figures

1 Introduction

Reasoning about actions and plans has always been a core issue in artifical intelligence research. Indeed, the underlying questions of causality, intension, and physical intuitions have a rich intellectual history in fields ranging from philosophy to physics, although this history has sometimes not been appreciated by researchers in AI.

Traditionally, planning has been seen as a form of problem-solving task. Given a specification of what is presently the case, an indication of the desired situation, and a description of the actions the agent can perform, the planner is to plot a course of action that leads from the current situation to the goal situation. The planner is effectively a "black box" that solves a particular class of problems, but that is inappropriate for or incapable of solving even closely related problems. In this respect, traditional planners are similar to expert systems with their deep but narrow knowledge. The main research issue to be addressed within this model is the combinatorics involved in the choices the planner can make.

Recently however, technological advances that have made computer systems and their interfaces more useable have led to interest in *mixed-initiative planning*. In a mixed-initiative system, several participants collaborate to develop and execute plans. Such a system might be an intelligent "assistant" such as those being envisoned to help people manage ever-increasing amounts of information, or it might be an autonomous agent that must take account of the actions of other agents and communicate with them to enable effective action. With more effective interfaces, including natural language and graphical displays, the traditional batch-oriented planner becomes a liability due to its relative inflexibility.

In any case, reasoning about actions and plans from a mixed-initiative perspective raises a variety of issues not addressed by the "black box" problem-solving model of traditional planning and the representations underlying it. This dissertation is an attempt to lay out the formal foundations of a knowledge representation suitable for mixed-initiative planning, in particular characterizing how it differs from the traditional model and offering new approaches to those issues. Although the emphasis here is on formal models, my aim is not solely a mathematical theory but also a functioning, "intelligent" system based on the theory. This pragmatic focus pervades the work reported here.

My investigation is in three parts.

1.1 Reasoning about Actions and Events

First, I consider the more fundamental problem of reasoning about actions in a given situation. That is, rather than being concerned with how action sequences are planned or recognized, we investigate how to represent temporally complex scenarios and predict or explain the consequences of possibly partial descriptions of the world. Although not directly concerned with mixed-initiative planning *per se*, the knowledge representation that I present, based on an explicit temporal logic with events, is motivated by the need to express the type of realistic, complex situations that arise in a communicative context, including but not limited to natural language communication.

Chapter 2 therefore begins with a recapitulation of previous work on the logical foundations of the interval temporal logic, including some new observations regarding reasoning effectively about change. Events are introduced next, followed by actions and a discussion of the causal reasoning that relates them.

A long-running thread in AI research about causality has to do with predicting the effects of actions (the *ramification* and *qualification* problems) and reasoning about their persistence (the *frame* problem). It is generally accepted that some form of assumption-based reasoning is required for such reasoning. My approach to these problems, foreshadowing later developments regarding plans, is to make whatever assumptions are necessary *explicit* rather than hiding them in the ontological assumptions of the model or in the semantic model theory. My motivation for this is that, first, it simplifies the semantics of the representation, and second, that it allows us to reason about the assumptions within the language as part of our reasoning about plans. The approach is an application of the *explanation closure* technique to the temporal logic, which I compare to other nonmonotonic approaches.

The technique is illustrated on a variety of problems based on the infamous Yale Shooting Problem. I believe that the solutions are both natural and reasonable, and their similarity across the suite of reasoning problems illustrates the scope of our approach. This is further evidenced by the ease with which the approach goes on to represent more complex situations involving external events and simultaneous, complex, interacting actions. Such phenomena are ubiquitous in the complex, realistic domains we want to plan in, yet are handled poorly, if at all, by many other representations of actions and time. Again, the ease with which such complex scenarios can be represented without grossly extending the original language strengthens the case for the event-based temporal logic as the foundation for realistic, natural planning and dialogue systems.

1.2 Plan Representation and Reasoning for Mixed-Initiative Planning

From the consideration of single scenarios, Chapter 3 shifts gears to consider mixed-initiative planning proper. I begin with an example of a mixed-initiative planning scenario recorded between two people, one playing the role of the "assistant" described

previously (a so-called "Wizard of Oz"scenario), the other playing the role of a "manager" with a task to accomplish. In addition to clearly demonstrating the breadth of knowledge required to build such a system, the example dialogue highlights several aspects of mixed-initiative planning that motivate my representation of plans and my approach to plan reasoning.

The first aspect is the *communicative* nature of mixed-initiative planning. This has several ramifications including the fact that planning and plan recognition (and other plan reasoning tasks) are continually interleaved during the interaction, and the need to refer to parts of plans and relationships between them in dialogue. I note that the communication need not be in natural language, and claim that equally complex phenomena arise from, for example, a well-designed graphical front end to a planning system or in a community of interacting planning agents. The second major aspect is that the reasoning being performed is fundamentally *defeasible*, that is subject to change given more information or more time to reason. The causes of this defeasibility include the fact that it is based on uncertain and incomplete knowledge, both of the world and, importantly, of other agents through their utterances or our observations, and the fact that the deliberation during plan reasoning is necessarily resource-bounded in interactive systems.

Based on these considerations, I propose a representation of plans based on an explicit system of defeasible reasoning that constructs *arguments* from *defeasible rules*. That is, plans are seen as arguments that a certain course of action under certain conditions will achieve certain goals. Planning is then a dialectical process involving the construction of arguments (plans) and the consideration of counter-arguments. The development of the approach includes a discussion of defeasible causal rules and the qualification problems and an application of the explanation closure technique to persistence reasoning and the frame problem. The argumentative, dialectical reasoning process is motivated by the communicative "give-and-take" of mixed-initiative planning. As I did for the representation of actions and time, I describe why traditional models of planning are inadequate for mixed-initiative planning, and in fact show how, in some cases, they can be reformulated as special cases of the more general method.

We are really only at the start of the study of mixed-initiative planning in AI. As a result, some of the proposals are incomplete and there are certainly many points at which different decisions could be made. I therefore conclude Chapter 3 with a discussion of some of these issues and with directions for future work. I are confident, however, that the representation of plans-as-arguments is appropriate for mixed-initiative planning.

1.3 The TRAINS-93 Domain Plan Reasoner

Finally, as at least a partial proof of concept for my work, the third and final part of the dissertation, Chapter 4, describes the implementation of these ideas as the domain plan reasoner of the TRAINS system. The TRAINS project is a long-term project whose goal is to develop an intelligent planning assistant that is conversationally proficient in natural language. Chapter 4 serves as both a description of the role of the domain

plan reasoner in the system as a whole, including design constraints imposed by being part of a bigger whole, and as a description of the implementation in detail, including the relationship between the program and the theories developed in the previous two chapters.

After a brief overview of the complete TRAINS system and a description of the role of the domain plan reasoner within it, we first describe the implementation of the underlying knowledge representation system based on the temporal logic representation of Chapter 2. We then describe the data structures used to represent plans based on the plans-as-arguments formalism of Chapter 3, and describe the algorithms that operate on them. We illustrate the functioning of the domain plan reasoner by presenting the TRAINS-93 sample dialogue in detail, including verbatim transcripts of the exchanges between the plan reasoner and the other modules of the system.

The TRAINS domain plan reasoner was intended to serve two somewhat contradictory goals. First, it embodies the theories described in the previous two chapters of the dissertation and provides evidence that they are at least plausible accounts of the phenomena. Of course, the mapping from theory to practice is not an exact one, and the discrepancies between the theories and the system need to be noted and then lead to further refinement of both. The second goal of the implementation was to function as part of actual running system being developed simulataneously by a large group of people. Although this imposed a number of pragmatic factors that affected the theoretical purity of the system, it also provided a much-needed incentive to develop a reasoning system with enough breadth to serve a variety of purposes. We believe that the current system will serve as a solid foundation for the development of the next generation of the TRAINS system, already under development, where the emphasis will be on filling out some of the depth of knowledge and reasoning that are missing from the current system.

2 Reasoning about Actions and Events[1]

This dissertation is concerned with knowledge representation and reasoning for mixed-initiative planning. But before we can reason about plans, we need to be able to represent the actions and events that make them up and reason about how they affect and are affected by the world. In this chapter, therefore, we present a representation of time, properties, actions, and events based on an explicit interval temporal logic. The approach is motivated by a need to express naturally the wide range of phenomena that arise in the description of realistic dynamic scenarios. Chapter 3 then develops a representation of plans based on a similarly broad perspective, and Chapter 4 describes our implementation of these ideas in an interactive planning system.

As motivation for this chapter, let us start by considering some of the properties of actions and events that we feel must be addressed by any general representation of them.

1. Actions and events take time. During this time, they may have a rich structure. For instance, the event of driving my car to work involves a wide range of different actions, states of the world and other complications, yet the activity over that stretch of time is appropriately described as a single event.

2. The relationship between actions and events and their effects is complex. Some effects become true at the end of the event and remain true for some time after the event. For example, when I put a book on the table, this has the effect that the book is on the table for at least a short time after the action is completed. Other effects only hold while the event is in progress—for example holding an elevator open by pressing the "open door" button. Other effects might start after the beginning of the event and end before it does, such as my having paid a toll while driving to work. This is an effect of the action even though it is not true at the end of it. Finally, it may be the case that the effects of actions are wholly independent of the action once the action is performed, as in a rock rolling down a hill after I nudge it out of place.

3. External changes in the world may occur no matter what actions an agent plans to do, and may interact with the planned actions. Possible external events should

[1]The work described in the chapter is based on a paper entitled "Actions and Events in Interval Temporal Logic," *Journal of Logic and Computation*, **4**(5), 1994. (Allen and Ferguson, 1994)

be an important factor when reasoning about what effects an action might have. Certain goals can only be accomplished by depending on external events, whether they are a result of natural forces (such as the wind blowing enabling sailing) or the actions of other agents (such as an engine arriving with some needed cargo).

4. Actions and events may interact in complex ways when they overlap or occur simultaneously. In some cases, they will interfere with certain effects that would arise if the events were done in isolation. In other cases the effects may be additive. And in still other cases, the effect of performing the two actions may be completely different from the effects of each in isolation.

5. Knowledge of the world is necessarily incomplete and unpredictable in detail, thus reasoning about actions and events can only be done on the basis of certain assumptions. No plan is foolproof, and it is important that a formalism makes the necessary assumptions explicit so that they can be considered in evaluating plans.

In addition, a general representation of actions and events must support the following somewhat overlapping tasks:

- Prediction: Given a description of a scenario, including actions and events, what will (or is most likely to) happen?

- Planning: Given an initial description of the world and a desired goal, find a course of action that will (or is most likely to) achieve that goal.

- Explanation: Given a set of observations about the world, find the best explanation of the data. When the observations are another agent's actions and the explanation desired is the agent's plan, and the problem is called plan recognition.

Throughout this chapter we will be comparing our approach based on interval temporal logic to more traditional AI models based on weaker temporal models, such as STRIPS, the situation calculus, and the event calculus. We will claim that these formalisms are either incapable of accomodating the issues listed above or, if they can be extended to do so, the extensions amount to the addition of a temporal logic to the original formalism. This chapter concentrates on the prediction and explanation problems, leaving planning for the next chapter.

This chapter proceeds as follows. Section 2.1 outlines our intuitions about actions and events and, in preparation for later comparisons, describes the two predominant AI models of action and change, the situation calculus and the STRIPS representation. Section 2.2 then presents the interval temporal logic in detail, beginning with the logic of time and then adding properties, events, and actions and considering their relationships. Section 2.3 concerns applying the formalism to reasoning about action in simple domains. We first describe our approach to the frame problem based on explicit explanation closure axioms, compare it to other nonmonotonic reasoning approaches, and then illustrate the method by solving a set of standard problems from the literature based on the Yale Shooting Problem. Although many of the problems are too simple to exercise the temporal logic, they do provide a convenient point of comparison with

other approaches. As well, the fact that solutions seem both natural and intuitive is evidence in favour of our approach. Finally, we look beyond the overly-simplified AI toy worlds and consider the implications of external events in Section 2.4 and of interacting simultaneous actions in Section 2.5. Both of these phenomena are ubiquitous in the realistic domains in which we want to plan. Since the work described in this chapter is part of a long tradition of work on the interval temporal logic representation, the chapter concludes with a summary of the approach in general and of the new contributions of this work in particular.

2.1 Representing Actions and Events

Before starting the formal development, we will attempt to describe the intuitions motivating the representation. We will then consider why the most commonly accepted representations of action in AI will not meet our needs.

2.1.1 Intuitions about Actions and Events

The first issue concerns what an event is. We take the position that events are primarily linguistic or cognitive in nature. That is, the world does not really contain events. Rather, events are the way by which agents classify certain useful and relevant patterns of change. As such, there are very few restrictions on what an event might consist of except that it must involve at least one object over some stretch of time, or involve at least one change of state. Thus the very same circumstances in the world might be described by any number of events. For instance, consider a circumstance in which a ball rolled off a table and bounced on the floor. This already is one description of what actually happened. The very same set of circumstances could also be described as the event in which the ball fell off the table, or in which the ball hit the ground, or in which the ball dropped. Each of these descriptions is a different way of describing the circumstances, and each is packaged as a description of an event that occurred. No one description is more correct than the other, although some may be more informative for certain circumstances, in that they help predict some required information, or suggest a way of reacting to the circumstances.

Of course, the "states" of the world referred to above are also ways of classifying the world, and are not inherent in the world itself either. Thus, the same set of circumstances described above might be partially described in terms of the ball being red. Given this, what can one say about the differences between events and states? Intuitively, one describes change and the other describes aspects that do not change. In language, we say that events occur, and that states hold. But it is easy to blur these distinctions. Thus, while the balling falling from the table to the floor clearly describes change and hence describes an event, what about the circumstance where an agent John holds the door shut for a couple of minutes. While the door remains shut during this time and thus doesn't change state, it seems that John holding the door shut is something that occured and thus is like an event. These issues have been studied extensively in work on the semantics of natural language sentences. While there are many proposals, everyone

agrees on a few basic distinctions (*e.g.*, (Vendler, 1967; Mourelatos, 1978; Dowty, 1986)). Of prime relevance to us here are sentences that describe *states* of the world, as in "The ball is red," or "John believes that the world is flat," and sentences that describe general ongoing *activities* such as "John ran for an hour," and *events* such as "John climbed the mountain." Each of these types of sentences has different properties, but the most important distinctions occur in their relation to time. All these sentences can be true over an interval of time. But when a state holds over an interval of time t, one can conclude that the state also held over subintervals of t. Thus, if the ball is red during the entire day, then it is also red during the morning. This property is termed *homogeneity* in the temporal logic literature. Events, on the other hand, generally have the opposite property and are anti-homogeneous: If an event occurs over an interval t, then it doesn't occur over a subinterval of t, as it would not yet be completed. Thus, if the ball dropped from the table to the floor over time t, then over a subinterval of t is would just be somewhere in the air between the table and floor. Activities, on the other hand, fall in between. They may be homogenous, as in the holding the door shut example above, but they describe some dynamic aspect of the world like events. This type of distinction must appreciated by any general purpose knowledge representation for action and change.

The other general point to make is that intervals of time play an essential role in any representation of events. Events are defined to occur over intervals of time, and cannot be reduced to some set of properties holding at instantaneous points in time. Of course, semantically, one can define time intervals as an ordered pair of time points, but the truth conditions must be defined in terms of these ordered pairs, and not in terms of the individual points. Specifically, if an interval were simply a set of points between two points, and truth over an interval was defined in terms of truth over all the points in the interval, then every predicate would have to be homogeneous.

Finally, a word on actions. The word "action" is used in many different senses by many different people. For us, an action refers to something that a person or robot might do. It is a way of classifying the different sorts of things than an agent can do to affect the world, thus it more resembles a sensory-motor program than an event. By performing an action, an agent causes an event to occur, which in turn may cause other desired events to also occur. For instance, I have an action of walking that I may perform in the hope of causing the event of walking to my car. Some theories refer to the event that was caused as the action, but this is not what we intend here. Rather, we will draw on an analogy with the robot situation, and view actions as programs. Thus, performing an action will be described in terms of running a program.

2.1.2 The Situation Calculus

The situation calculus means different things to different researchers. In its original formulation (McCarthy and Hayes, 1969), which we will call the general theory of the situation calculus, situations are introduced into the ontology as a complete snapshot of the universe at some instant in time. In effect, the situation calculus is a point-based temporal logic with a branching time model. In its most common use, which we will call

the constructive situation calculus, it is used in a highly restricted form first proposed by Green (1969), in which the only way situations are introduced is by constructing them by action composition from an initial state. This practice has attracted the most attention precisely because the formalism is constructive—specifically, it can be used for planning. To contruct a plan for a goal G, prove that there exists a situation s in which G holds. In proving this, the situation is constructed by action composition, and thus the desired sequence of actions (the plan) can be extracted from the proof. As others have pointed out (*e.g.*, (Schubert, 1990)), most of the criticisms about the expressibility of the situation calculus concern the constructive form of it rather than the general theory. Our position is that the constructive situation calculus is a limited representation, especially in dealing with temporally complex actions and external events. The general theory, on the other hand, is much richer and can be extended to a model much closer to what we are proposing, but at the loss of its constructive aspect.

To see some of the difficulties, first consider a very simple example. In a domain that reasons about transportation of cargo, we might want to define an action of a train moving some cargo between two cities. Some of the information needed to reason about this action is the following:

1. The train initially starts at the originating city S.
2. The trip typically takes between 4 and 6 hours.
3. The cars must remain attached to the train during the entire trip.
4. The train will be on track segment $A1$ then it will cross junction $J1$, and be on track segment $A2$ for the rest of the trip.
5. The train will end up at the destination city D.

This is all very mundane information, but each fact might be crucial for some planning task. For instance, knowledge of the track segments that the train is on during the trip might be used to avoid having multiple trains on the same track at the same time.

In its constructive form, the situation calculus can only adequately represent properties (1) and (5). Actions are represented syntactically as functions from one situation to the resulting situation, and there appears to be no mechanism for representing information about the duration of actions (as needed for property 2), for representing effects that do not start at the end of the action (property 4), or for representing preconditions that must hold beyond the start time of the action (property 3). In addition, this example didn't include the more difficult problems of representing external events, or interacting simultaneous actions. While there is now a considerable literature on extensions to the formalism to handle some of these problems, most lose the constructivity property and thus become less suitable for practical reasoning tasks such as planning. In addition, most of the extensions involve syntactic extensions to the language that are defined semantically by particular minimization strategies specifically tailored to the problem. As a result, trying to combine solutions to all the problems into a single uniform formalism is remarkably complex, even when remaining focused on carefully

hand-tailored sample problems. The chances of integrating such representations into a more general knowledge representation system for tasks such as real-world planning or natural language understanding seem remote.

2.1.3 The STRIPS Representation

The STRIPS representation (Fikes and Nilsson, 1971) adds additional constraints to the situation calculus model and is used by most implemented planning systems built to date. In STRIPS, a state is represented as a finite set of formulas, and the effects of an action are specified by two sets of formulas: the delete list specifies what propositions to remove from the initial state, and the add list specifies the new propositions to add. Together, they completely define the transition between the initial state and the resulting state. Figure 2.1 shows a simple Blocks World action that involves placing one block on top of another (the `STACK` action). The preconditions on an action indicate when the action is applicable—in this case it is applicable whenever both blocks are clear. The effects state that one block is now on top of the other (the add list) and that the bottom block is not clear (the delete list). The operation for constructing a resulting state applies the delete list first and then asserts the add list.

Like the situation calculus, STRIPS-style actions are effectively instantaneous and there is no provision for asserting what is true while an action is in execution. Also, since the state descriptions do not include information about action occurrences, such systems cannot represent the situation where one action occurs while some other event or action is occurring. The validity of the method comes from the STRIPS assumptions, which assume that the world only changes as the result of a single action by the agent, and that the action definitions completely characterize all change when an action is done. Of course, these assumptions would not be valid if simultaneous actions or external events were allowed.

While discussing the world representation in such models, it is important to include explicitly the nonlinear planners (*e.g.*, (Sacerdoti, 1975; Tate, 1977; Vere, 1983; Chapman, 1987; Wilkins, 1988; McAllester and Rosenblitt, 1991)) as the representational power of these systems is often misunderstood. In particular, the underlying world model used by such systems is essentially the same state-based model as used in STRIPS. For example, Figure 2.2 shows a very simple nonlinear plan. It represents a partial ordering of actions, where action A must precede B and C, and B and C must precede action D, but actions B and C are unordered with respect to each other. This is actually a compact representation of two distinct linear plans, namely, A, B, C, D in order, or A, C, B, D in order. It does not include the possibility that actions B and C are simultaneous. Nonlinearity is a property of the search process in constructing plans, and not a property of the representation of the world.

Nonlinear planning was developed so that decisions about action ordering could be delayed as long as possible, avoiding the need for backtracking in cases where it was not necessary. But information about a particular state can only be obtained by inferring what remains true throughout all the possible linear orderings. Thus, if we look at the example in Figure 2.2, the only assertions the system could make about the state after

```
STACK(a,b)
        preconds:  (Clear a) (Clear b)
        delete:    (Clear b)
        add:       (On a b)
```

Figure 2.1: Actions as state change in STRIPS

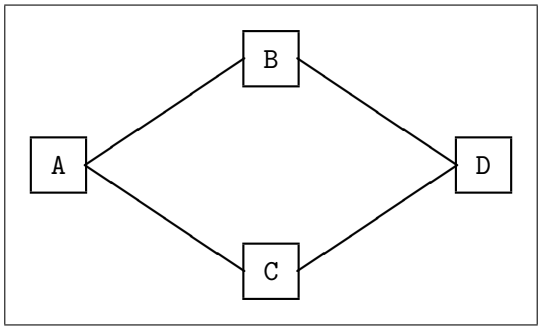


Figure 2.2: A simple nonlinear plan

D would be those assertions that were true both in the state obtained by applying A, B, C, and D in sequence *and* in the state obtained by applying A, C, B, and D in sequence. For example, if B were "paint the room blue," and C were "paint the room crimson," then the color of the room after D would be undetermined, since the final color of the room given order A, B, C, D would be crimson, whereas the color of the room given the order A, C, B, D would be blue. If, instead, B were "paint the chair blue," then the result of the nonlinear plan would be that the room is crimson (from C) and the chair is blue (from B), since this is the resulting state given either the ordering A, B, C, D or the ordering A, C, B, D.

This final example suggests an extension to handle some cases of simultaneous action. In particular, in the last example above, the effect of performing the two actions B and C simultaneously could be the same as the result computed for any of the action orderings allowed by the nonlinear plan. Tate (1977), Vere (1983), and Wilkins (1988) have all used this extension in their systems and allow simultaneous action when the two actions are independent of each other. In such cases, the effect of the two acts performed together is the simple union of the individual effects of the acts done in isolation. Wilkins develops an interesting approach using the notion of resources to reason about the independence of actions. The problem with this solution is that it excludes common situations of interest in realistic domains, namely where simultaneous actions are not independent, such as when they have additional synergistic effects or interfere with each other.

The STRIPS assumptions appear to be fundamentally incompatible with interacting simultaneous actions and external events. They hold only in worlds with a single agent, who can only do one thing at a time, and where the world only changes as the result of the agent's actions. However, they are so ingrained in the planning literature that many researchers don't even acknowledge their limitations. In some sense, they have become part of the definition of the classical planning problem.

2.2 Interval Temporal Logic

Having described our motivations, we now start the development of the temporal logic. We start by describing the basic temporal structure to be used in the logic, namely the interval representation of time developed by Allen (1983a; 1984) and discussed in detail in (Allen and Hayes, 1989). We then describe the temporal logic used to represent knowledge of properties, events, and actions. We conclude this section with a comparison of related formalisms. Subsequent sections will explore how the representation supports reasoning about events and actions, especially in complex situations with external events and simultaneous actions.

2.2.1 The Structure of Time

The temporal structure we assume is a simple linear model of time. Notions of possibility that are introduced in branching time models or the situation calculus would

be handled by introducing a separate modal operator to represent possibility explicitly. Since such a modal operator is needed in general anyway, there seems no need to build it into the temporal structure. The temporal theory starts with one primitive object, the time period, and one primitive relation: $Meets$.

A time period intuitively is the time associated with some event occuring or some property holding in the world. Intuitively, two periods m and n meet if and only if m precedes n, yet there is no time between m and n, and m and n do not overlap. The axiomatization of the $Meets$ relation is as follows, where i, j, k, l, and m are logical variables restricted to time periods. The axioms are presented graphically in Figure 2.3. First, there is no beginning or ending of time and there are no semi-infinite or infinite periods. In other words, every period has a period that meets it and another that it meets:

$$\forall i \,.\, \exists j, k \,.\, Meets(j, i) \wedge Meets(i, k). \tag{2.1}$$

Second, periods can compose to produce a larger period. In particular, for any two periods that meet, there is another period that is the "concatenation" of them. This can be axiomatized as follows:

$$\forall i, j, k, l . Meets(i, j) \wedge Meets(j, k) \wedge Meets(k, l) \supset \exists m . Meets(i, m) \wedge Meets(m, l). \tag{2.2}$$

As a convenient notation, we will often write $j + k$ to denote the interval that is the concatenation of intervals j and k. This functional notation is justified because we can prove that the result of $j + k$ is unique (Allen and Hayes, 1989).

Next, periods uniquely define an equivalence class of periods that meet them. In particular, if i meets j and i meets k, then any period l that meets j must also meet k:

$$\forall i, j, k, l \,.\, Meets(i, j) \wedge Meets(i, k) \wedge Meets(l, j) \supset Meets(l, k). \tag{2.3}$$

These equivalence classes also uniquely define the periods. In particular, if two periods both meet the same period, and another period meets both of them, then the periods are equal:

$$\forall i, j, k, l \,.\, Meets(k, i) \wedge Meets(k, j) \wedge Meets(i, l) \wedge Meets(j, l) \supset i = j. \tag{2.4}$$

Finally, we need an ordering axiom. Intuitively, this axiom asserts that for any two pairs of periods, such that i meets j and k meets l, then either they both meet at the same "place," or the place where i meets j precedes the place where k meets l, or vice versa. In terms of the meets relation, this can be axiomatized as follows, where the symbol "$\otimes$" means "exclusive-or":

$$\begin{aligned} \forall i, j, k, l \,.\, (Meets(i, j) \wedge Meets(k, l)) \supset \\ Meets(i, l) \otimes (\exists m \,.\, Meets(k, m) \wedge Meets(m, j)) \otimes \\ (\exists m \,.\, Meets(i, m) \vee Meets(m, l)). \end{aligned} \tag{2.5}$$

Many of the properties that are intuitively desired but have not yet been mentioned are actually theorems of this axiomatization. In particular, it can be proven that no

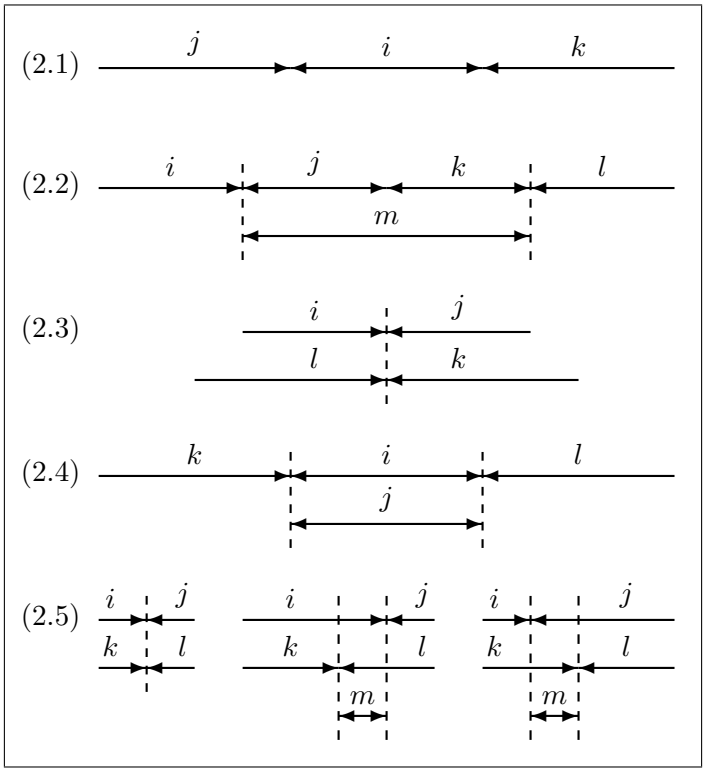


Figure 2.3: The axiomatization of time periods

period can meet itself, and that if one period i meets another j then j cannot also meet i (*i.e.*, finite circular models of time are not possible).

With this system, one can define the complete range of the intuitive relationships that could hold between time periods. For example, one period is before another if there exists another period that spans the time between them, for instance:

$$Before(i, j) \equiv \exists m \,.\, Meets(i, m) \wedge Meets(m, j).$$

Figure 2.4 shows each of these relationships graphically (equality is not shown). We will use the following symbols to stand for commonly-used relations and disjunctions of relations:

$Meets(i, j)$	$i : j$
$Before(i, j)$	$i \prec j$
$Before(i, j) \vee Meets(i, j)$	$i \prec\!: j$
$During(i, j) \vee Starts(i, j) \vee Finishes(i, j)$	$i \sqsubset j$
$During(i, j) \vee Starts(i, j) \vee Finishes(i, j) \vee i = j$	$i \sqsubseteq j$

Finally, an important relationship between two periods is *Disjoint*: two intervals are disjoint if they do not overlap in any way. We write this as "$i \bowtie j$" and define it by

$$i \bowtie j \equiv i \prec\!: j \vee j \prec\!: i.$$

The computational properties of the interval calculus and algorithms for maintaining networks of temporal constraints are presented in (Allen, 1983a; Vilain and Kautz, 1986; Vilain et al., 1990).

A period can be classified by the relationships that it can have with other periods. For example, we call a period that has no subperiods (*i.e.*, no period is contained in it or overlaps it) a *moment* and a period that has subperiods, an *interval*. In addition, we can define a notion of time point by a construction that defines the beginning and ending of periods. It is important to note that moments and points are distinct and cannot be collapsed. In particular, moments are true periods and may meet other periods, and if $Meets(i, m) \wedge Meets(m, j)$ for any moment m, then i is before j. Points, on the other hand, are not periods and cannot meet periods. Full details can be found in (Allen and Hayes, 1989).

Any semantic model that allows these distinctions would be a possible model of time. In particular, a discrete time model can be given for this logic, where periods map to pairs of integers $\langle I, J \rangle$ where $I < J$. Moments correspond to pairs of the form $\langle I, I + 1 \rangle$, and points correspond to the integers themselves. A similar model built out of pairs of real numbers does not allow moments. A more complex model can be specified out of the reals, however, that does allow continuous time, or models that are sometimes discrete and sometimes continuous are possible. Ladkin and Maddux (Ladkin and Maddux, 1988) have characterized the set of possible models as precisely the arbitrary unbounded linear orders.

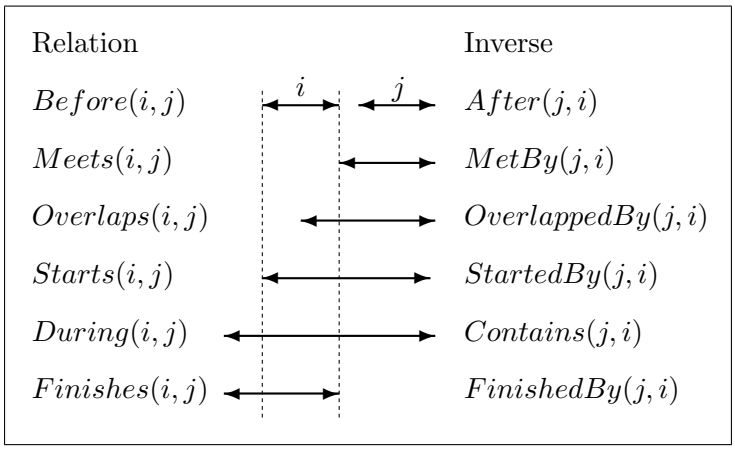


Figure 2.4: The possible relations between time periods (equality not shown)

2.2.2 Interval Temporal Logic

The most obvious way to add times into a logic is to add an extra argument to each predicate. For example, in a nontemporal logic, the predicate *Green* might denote the set of green objects. Thus, a formula such as $Green(frog13)$ would be true only if the object named by the term, $frog13$, is in the set of green objects. To make this a temporal predicate, a time argument is added and *Green* now denotes a set of tuples consisting of all green objects with the times at which they were green. Thus, the proposition $Green(frog13, t1)$ is true only if the object named by $frog13$ was green over the time named by $t1$.

By allowing time intervals as arguments, we open the possibility that a proposition involving some predicate P might be neither true nor false over some interval t. In particular, consider a predicate P that is true during some subinterval of t and also false in some other subinterval of t. In this case, there are two ways we might interpret the negative proposition $\tilde{}P(t)$. In the weak interpretation, $\tilde{}P(t)$ is true if and only if it is not the case that P is true throughout interval t, and thus $\tilde{}P(t)$ is true if P changes truth-values during t. In the strong interpretation of negation, $\tilde{}P(t)$ is true if and only if P is false throughout t, and so neither $P(t)$ nor $\tilde{}P(t)$ would be true in the above situation. Thus, a logic with only strong negation has truth gaps.

We take the weak interpretation of negation as the basic construct, as do Shoham (1987) and Bacchus *et. al.* (1989), to preserve a simple two-valued logic. Weak negation also seems to be the appropriate interpretation for the standard definition of implication. In particular, the formula $P(t) \supset Q(t')$ is typically defined as $\tilde{}P(t) \vee Q(t')$. Since we want the implication to mean "whenever P is true over t, then Q is true over t'," this is best captured by the weak negation form of the equivalent formula. With weak negation, $\tilde{}P(t) \vee Q(t')$ says that either P is not true throughout t (but might be true in some subintervals of t), or Q is true over t'. This seems the right interpretation. Of course, we can still make assertions equivalent to the strong negation. The fact that P is false throughout t can be expressed as

$$\forall t' \,.\, t' \sqsubseteq t \supset \tilde{}P(t').$$

This is a common enough expression that we will introduce an abbreviation for the formula, namely $\neg P(t)$ (that is, the symbol "$\neg$" means strong negation). This way we obtain the notational convenience of strong negation while retaining the simpler semantics of a logic with no truth gaps.

There are several characteristics of propositions that allow them to be broadly classified based on their inferential properties. These distinctions were originally proposed by Vendler (1967), and variants have been proposed under various names throughout linguistics, philosophy, and artificial intelligence ever since (*e.g.*, (Mourelatos, 1978; Allen, 1984; Dowty, 1986; Shoham, 1988)). For the most part we will not be concerned with all the distinctions considered by these authors. However one important property mentioned previously is homogeneity. Recall that a proposition is homogeneous if and only if when it holds over a time period t, it also holds over any period within t. In the current formulation, this property is defined by a family of axiom schemata, one for each arity of predicate. For all homogeneous predicates P of arity $n + 1$:

Homogeneity Axiom Schema

$$\forall x_i, t, t' \,.\, P(x_1, \ldots, x_n, t) \wedge t' \sqsubseteq t \supset P(x_1, \ldots, x_n, t').$$

All predicates will be homogeneous in what follows except when explicitly noted. The following useful theorem follows from the definition of strong negation and the homogeneity property, for any homogeneous predicate P:

DISJ $\forall t, t' \,.\, P(t) \wedge \neg P(t') \supset t \bowtie t'.$

That is, two intervals over which a predicate has different truth values (with respect to strong negation) must be disjoint.

The use of weak negation as the basic construct might also allow intervals over which $\sim P$ holds but for no subinterval of which does $\neg P$ hold. Within such an interval, there would be an infinite telescoping sequence of intervals within which P alternately holds and does not hold. We rule out such intervals with the following axiom (schema) of discrete variation:

Discrete Variation Axiom Schema

$$\forall t \,.\, \sim P(t) \supset \exists t' \,.\, t' \sqsubseteq t \wedge \neg P(t').$$

Hamblin (1972) refers to this axiom and the homogeneity axiom as characterizing the "phenomenal" predicates. Further discussion of discrete variation and its relation to infinite series in calculus and physics can be found in (Davis, 1992).

Finally, combining the axiom of discrete variation with the homogeneity axioms allows us to derive the following theorem useful for reasoning about when properties change truth values:

TRPT $\forall t, t' \,.\, P(t) \wedge \neg P(t') \wedge t \prec: t' \supset$
$\exists T, T' \,.\, P(T) \wedge \neg P(T') \wedge T : T' \wedge t' \sqsubseteq T'$

Defining a *transition point* to be the existence of adjacent intervals for which a property has complementary truth values, this axiom states that if a property has changed truth value, then there must be transition point where it does so.

2.2.3 The Logic of Events

The logic developed thus far is still insufficient to conveniently capture many of the circumstances that we need to reason about. In particular, we need to introduce events as objects into the logic. There are many reasons for this, and the most important of these are discussed in the remainder of this section.

Davidson (1967) argued that there are potentially unbounded qualifications that could be included in an event description. For example, the event of Jack lifting a particular ball might be asserted to occur at some time by a predicate LIFT, as

$\textsc{Lift}(t1, jack34, ball26)$. The problem now arises in representing the event "Jack lifted the ball onto the table." Either we need to add another argument to the LIFT predicate, or we need to introduce a new predicate that represents a variant of lifting that includes an extra argument. Neither is satisfactory. In the former case, all predicates describing event occurrences will contain a large number of argument positions, typically unspecified in any particular event description and thus requiring a large number of existential variables. In the latter case, we have a large class of predicates all asserting essentially the same thing. If we could put a bound on the number of argument positions needed, then one of these solutions might be viable. But in fact, it seems we could always add additional information about the event that has not yet been captured by a parameter, forcing us to add another. In natural language, this creates particular difficulty when representing adverbial modifiers. Davidson suggested the solution of reifying events, whereby additional modifiers would simply become additional predications on the event. Thus, the event of Jack lifting the ball onto the table with the tongs might be represented as

$$\exists e \,.\, \textsc{Lift}(e, t1, jack34, ball26) \wedge dest(e) = table5 \wedge instrument(e) = tongs1.$$

The issue of reifying events is not only an issue for representing natural language meaning, however. A sophisticated plan reasoning system also needs to represent and reason about events of similar complexity. In addition, in many forms of plan reasoning, the system must be able to distinguish events even though it does not have any information to distinguish them. For instance, in a plan recognition task, an agent might know that two separate lifting events occurred, but it knows little else. In a reified logic this is easy to state, namely, as

$$\exists e_1, e_2 \,.\, \textsc{Lift}(e_1) \wedge \textsc{Lift}(e_2) \wedge e_1 \neq e_2.$$

However, in a nonreified logic such a situation would have to be represented by a complex formula growing in size with the number of arguments. Thus, if the LIFT predicate took five arguments, we would need a formula such as

$$\begin{aligned}\exists t_1, t_2 \,.\, \forall a_1, \ldots, a_8 \,.\, \textsc{Lift}(t_1, a_1, a_2, a_3, a_4) \wedge \textsc{Lift}(t_2, a_5, a_6, a_7, a_8) \wedge \\ (t_1 \neq t_2 \vee a_1 \neq a_5 \vee a_2 \neq a_6 \vee a_3 \neq a_7 \vee a_4 \neq a_8).\end{aligned}$$

Besides being cumbersome, this formulation is committed to always being able to distinguish lifting events on the basis of the five arguments given. If Davidson's argument is accepted and the number of qualifications could be unlimited, we might not be able to express the situation at all.

We use a representation that makes event variables the central component for organizing knowledge about events. In particular, events are divided into types, and each type defines a set of role functions that specify the arguments to any particular event instance. The event of Jack lifting the ball onto the table at time $t1$ would be represented as

$$\begin{aligned}\exists e \,.\, \textsc{Lift}(e) \wedge time(e) = t1 \wedge \\ agent(e) = jack34 \wedge dest(e) = table5 \wedge theme(e) = ball26.\end{aligned}$$

Event predicates will always be written in SMALL CAPS to distinguish them from other functions and predicates.[2]

This representation is somewhat verbose for presenting examples. When we need to make a point, the representation using only functions on events will be used. At other times, however, we will use the more standard predicate-argument notation as a convenient abbreviation. Thus, we will usually abbreviate the above formula as:

$$\exists e \,.\, \textsc{Lift}(e, t1, jack34, ball26, table5).$$

The arguments in this predicate-argument form will depend on the predicate, but the first two argument positions will always be the event instance and the time of the event, respectively. Finally note that event predicates are anti-homogeneous (that is, they hold over no sub-interval of the time over which they hold) as discussed in Section 2.1.1.

Because the representation of events is based on role functions, an event instance uniquely defines all its arguments. This is important to remember when the predicate-argument abbreviation is used. For example, if we asserted that both $\textsc{Lift}(e, t_1, a_1, b_1, c_1)$ and $\textsc{Lift}(e, t_2, a_2, b_2, c_2)$ were true, then this would entail that $a_1 = a_2$, $b_1 = b_2$, $c_1 = c_2$, and $t_1 = t_2$, a fact not obvious when using the predicate-argument form but clear from the functional form.

We will represent knowledge about events in several ways. The first is by defining necessary conditions on the event occurring. For instance, consider the event of one block being stacked on another by a robot. This could be described by an event predicate of form $\textsc{Stack}(e, t, x, y)$. Axioms then define the consequences of this event occurring. For instance, one might assert that whenever a stack event occurs, the first block is on the second block at the end of the action. In the predicate-argument notation, this could be written as

$$\forall e, t, x, y \,.\, \textsc{Stack}(e, t, x, y) \supset \exists t' \,.\, t : t' \wedge On(x, y, t').$$

In the functional form, the same axiom would be

$$\forall e \,.\, \textsc{Stack}(e) \supset \exists t' \,.\, time(e) : t' \wedge On(block1(e), block2(e), t').$$

Of course, there are many other necessary conditions in order for a stacking event to occur. For instance, we might say that the block moved ($block1(e)$) must be clear when the event starts, that the agent must be holding that block some time during the event (actually up to the end of the event), and that the other block ($block2(e)$) is clear just before the end of the event, and has *block1* on it immediately after the event completes.

[2]The logic containing reified events also allows one to discuss events that do not occur. In particular, asserting that an event instance exists does not necessarily entail that it occurred. A new predicate *Occurs* can be introduced and the assertion that Jack lifted the ball (at time $t1$) would be represented as $\exists e \,.\, \textsc{Lift}(e, t1, jack34, ball26) \wedge Occurs(e)$. Using an explicit *Occurs* predicate allows events to exist even if they do not actually occur, or could never occur. We will not need this expressive power for what follows and so will proceed under the interpretation that only events that occur exist. Not having to put the *Occurs* predicate in each formula will simplify the presentation.

The event terminates at the time when *block1* is on *block2*. This information can be expressed directly using the temporal logic by the following axiom:

$$\begin{aligned}
&\forall e, t, x, y \,.\, \textsc{Stack}(e, t, x, y) \supset \\
&\quad \exists j, k, l, m, n \,.\, Clear(x, j) \wedge Overlaps(j, t) \wedge \\
&\qquad Holding(x, k) \wedge Finishes(k, t) \wedge j : k \wedge \\
&\qquad Clear(x, l) \wedge t : l \wedge Clear(y, m) \wedge SameEnd(t, m) \wedge \\
&\qquad On(x, y, n) \wedge t : n.
\end{aligned}$$

A more useful form of this axiom for planning uses temporal functions on each event that define the structure of the temporal intervals needed for its definition. For example, for the class of stacking events, we need functions to produce times corresponding to the existential variables in the axiom given above. Using new function names, we might define the temporal structure of the stacking event as follows:

$$\begin{aligned}
&\forall e, t \,.\, \textsc{Stack}(e, t) \supset \\
&\quad Overlaps(\mathit{pre1}(e), t) \wedge Finishes(\mathit{con1}(e), t) \wedge \mathit{pre1}(e) : \mathit{con1}(e) \wedge \\
&\quad t : \mathit{eff1}(e) \wedge SameEnd(t, \mathit{pre2}(e)) \wedge t : \mathit{eff2}(e).
\end{aligned}$$

The temporal functions are named to informally suggest the three classes of conditions that arise in an event definition. The "preconditions"—conditions that must hold prior to the event's occurrence—have the prefix *pre*, the "effects"—conditions that must hold following the event—have the prefix *eff*, and the other conditions that must hold during the event have the prefix *con*. The temporal functions can be viewed as Skolem functions justified by the original axiom. The only aspect that will seem puzzling is that we will use the same Skolem functions in several axioms. This should be viewed as a notational convenience that allows a large conjunctive axiom (with a single universally quantified event variable) to be presented as several smaller axioms. With this temporal structure defined for every stacking event, the axiom defining the necessary conditions for the event's occurrence now can be expressed as follows:

$$\begin{aligned}
&\forall e, t, x, y \,.\, \textsc{Stack}(e, t, x, y) \supset \\
&\quad Clear(x, \mathit{pre1}(e)) \wedge Holding(x, \mathit{con1}(e)) \wedge Clear(x, \mathit{eff1}(e)) \wedge \\
&\quad Clear(y, \mathit{pre2}(e)) \wedge On(x, y, \mathit{eff2}(e)).
\end{aligned}$$

This axiom is shown graphically in Figure 2.5.

The above axiom asserts what is true whenever a stacking event occurs, independent of the situation. Other knowledge about events is relevant only in certain situations (*i.e.*, the event has conditional effects). For instance, if the block being moved in a stacking action was initially on another block, then this other block becomes clear. This is expressed in the logic by the following axiom, which states that if block x was initially on another block z, then z becomes clear when x is moved:

$$\begin{aligned}
&\forall e, t, x, y, z, t' \,.\, \textsc{Stack}(e, t, x, y) \wedge On(x, z, t') \wedge Overlaps(t', t) \supset \\
&\qquad \exists T, t'' \,.\, t' \sqsubseteq T \wedge On(x, z, T) \wedge T : \mathit{con1}(e) \wedge Clear(z, t'') \wedge T : t''.
\end{aligned}$$

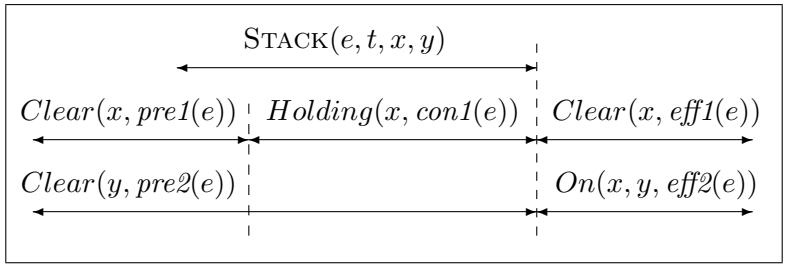


Figure 2.5: Necessary conditions for stacking x on y

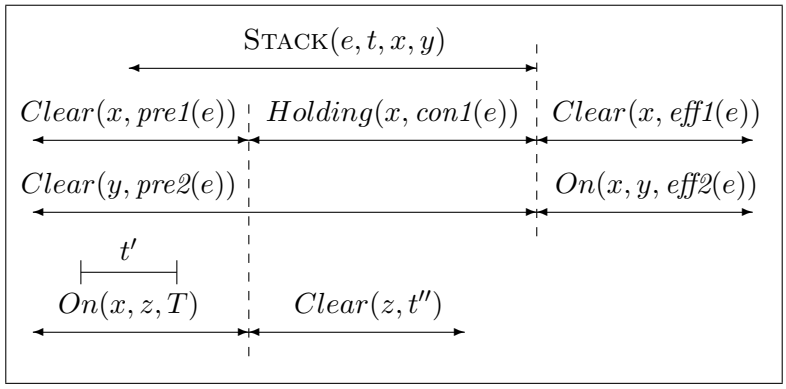


Figure 2.6: Conditional effects of stacking when x is initially on z

This situation is illustrated graphically in Figure 2.6. This axiom applies in a situation with three blocks, such as A, B, and C, where A is originally on block C. Note that this definition does not assert that block C will be clear at the end of the stacking event. In particular, if two stacking events overlap in time (say, $\textsc{Stack}(e1, t1, A, B)$ and $\textsc{Stack}(e2, t2, D, C)$), then this may not be the case, for D may be placed on C before A is placed on B. Most other representations cannot easily represent such subtleties.

It is worth noting that the conditional effect axiom above, although complicated by the interval-based ontology, does in fact demonstrate a typical pattern. Notice that the left-hand side of the implication contains the proposition $On(x, z, t')$, while the right-hand side contains $On(x, z, T)$ where $t' \sqsubseteq T$. The new interval T defines the extent of the property given that the event has occurred. While this formulation seems a bit cumbersome, it occurs with every conditional effect. Thus it would be easy to define an abbreviation convention to make specifying such axioms more convenient.

Of course, if the only events we needed to represent were simple events such as occur in the blocks world, then the temporal logic would be overkill. But we are interested in much more realistic events, where we may have significant knowledge about how the world changes as the event occurs. As a quick example, consider the event of a cup filling up with water. At the end the cup is full, but while the event is occurring, the level of the water is continually increasing. Commonsense knowledge about this event is easily captured in the interval logic. For instance, the fact that the level of the water continues to rise throughout the event is captured by the following axiom:

$$\begin{aligned}\forall e, t, c, t', t'' \,.\, &\textsc{Fill}(e, t, c) \supset \\ &\forall t, t'' \,.\, (t' \sqsubset t) \wedge (t'' \sqsubset t) \wedge (t' \prec: t'') \supset \\ &\quad level(c, t') < level(c, t''),\end{aligned}$$

where $level(c, t)$ is a function that gives the minimum level of the water in the cup over time t. This axiom captures the intuition that the water continually rises quite simply and directly without making a commitment to a continuous model of change. Dealing with events that involve such "continuous change" is not the focus of this work, however, and most of our examples will involve the very simple actions that are common in the literature on planning and reasoning about action.

Finally, some might wonder why the logic has no separate mechanism for describing processes. While this may ultimately be necessary, the existing formalism already supports a wide range of "process-like" predicates. In particular, you can define a property that is true only when an event is occurring. In the above example about the cup, we could define a predicate $CupFilling$ as

$$\forall c, t \,.\, CupFilling(c, t) \equiv \exists t', e \,.\, t \sqsubseteq t' \wedge \textsc{Fill}(e, t', c)$$

Note that by this definition, $CupFilling$ is a homogeneous predicate, as expected for properties. The place where problems arise with this simple approach is in dealing with the imperfective paradox. For instance, you might want to say that a person was crossing the street at some time t, even if they changed their mind at the last minute and went back to the original side. This problem has been studied extensively in the

philosophical literature, but has not been a focus for our work as it does not seem to arise naturally in the planning domains we have studied.

2.2.4 The Logic of Actions

The representation of events described in the previous section cannot adequately capture knowledge of causality. In particular, the formulas above do not state what properties are caused by the stacking action or what properties simply must be true whenever the action succeeds. This is the distinction that STRIPS makes between preconditions and effects. Intuitively, it is evident that the stacking action causes block A to be on block B in situations where both blocks are clear at the start of the action. Furthermore, the stacking action causes block B to become not clear while it does not affect the condition that block A is clear.

To encode such knowledge, we need to be able to reason about attempting to perform an action. The logic developed so far can express the fact that a certain event occurred, but not that an agent attempted to perform some action. Until we make such a distinction, we will not be able to explicitly describe the conditions under which an action can be successfully executed, or describe what happens when an action is tried in inappropriate circumstances. To do this, we need to better define the distinction between events and actions. So far, we have only talked about events occurring. The assertion that Jack lifted the ball onto the table describes an event in which Jack performed some action that resulted in the ball being lifted onto the table. The action Jack performed, namely, the lifting, corresponds to some set of motions that Jack performed in order to lift the ball. If Jack were a robot, the action would be the execution of a program that involved the correct control sequences given perceptual input. Thus, in a robot world the action corresponds to the program, whereas the event corresponds to a situation in which the program was executed successfully.

As noted in Section 2.1.1, for every action there is a corresponding event consisting of an agent performing that action. We will often exploit this by using the same names for events and actions. Thus the $\textsc{Stack}(e, t, x, y)$ predicate presented in the previous section might correspond to the action term $stack(x, y)$ that denotes a program where x and y correspond to the blocks being stacked. Of course, there are other events that do not involve actions. For example, natural forces (such as the wind blowing) result in events but do not involve action in the sense we are using it. Actions may be arbitrarily complex activities, and can be decomposable into other less complex actions, which themselves may be decomposable, until a certain basic level of action is attained. The primitive actions are called the *basic* actions following (Goldman, 1970).

The predicate Try is defined on programs, such that $Try(\pi, t)$ is true only if the program π is executed over time t.[3] As with event predicates, Try is anti-homogeneous. We can now assert an axiom defining the conditions sufficient to guarantee successful action attempts. In the case of stacking, whenever the agent tries to stack x on y

[3]In previous work (*e.g.*, (Allen, 1991b)), we included the event "caused" by attempting the action as an argument to the Try predicate. The problem with this is that we might want to categorize the "caused" event in several ways (*i.e.*, using several event predicates).

starting in a situation where x and y are clear, then a stacking event occurs that is temporally constrained by the initial conditions:

$$\forall x, y, t, j, k \,.\, Try(stack(x, y), t) \wedge Clear(x, j) \wedge Overlaps(j, t) \wedge$$
$$Clear(y, k) \wedge SameEnd(k, t) \supset \exists e \,.\, \textsc{Stack}(e, t, x, y)$$

Of course, a realistic axiom would also include duration constraints on t so that the action is attempted sufficiently long enough to allow it to succeed, *etc.*.

Note that the logic does not presuppose any particular relationship between when the action is performed and when the event caused occurs. The only constraint we need, based on commonsense notions of causality, is that the caused event cannot precede the action that caused it. But otherwise, they can be simultaneous, or the event might be included within the time of the action performance, or it might immediately follow the action. In practice, there are two forms of axioms that are most common and useful. In *simultaneous* causality, the action and the event caused are simultaneous, as in the above example in which the stacking event is co-temporal with the action execution (recall that the abbreviation $\textsc{Stack}(e, t, a, b)$ implies $t = time(e)$). With *pushbutton* causality, the event immediately follows the action, say as the action of pushing a ball causes it to roll down a hill. Note that the issue of how the action relates to the event is independent of how an event relates to its effects, although the same distinctions can be made. For example, the effect of stacking two blocks starts holding at the end of the event. The effect of holding a door shut, on the other hand, might only hold while the event is occuring. Which relationships between actions, events and effects, is best to use for any particular problem will depend on one's intuitions about the domain, and the level of detail at which one needs to reason about the domain.

It is important to consider why the information about stacking is captured by two different sets of related axioms: one capturing the necessary conditions whenever a stacking event occurs (Section 2.2.3), and the other relating action attempts to event occurrences (above). This is because the two sets of axioms represent two very different sources of knowledge. The first defines knowledge about what the world is necessarily like whenever the event occurs successfully, while the second defines the abilities of the agent in causing events. In many situations, an agent may know the former but not the latter. For example, we all can recognize that the mechanic fixed our car, even if we have no idea what enabled the mechanic to do the job. In this case, we have knowledge of what it means to fix a car as expressed by the following axiom, which states that fixing something means that the thing was once broken and is now working:

$$\forall e, t, c \,.\, \textsc{Fix}(e, t, c) \supset$$
$$Broken(c, pre1(e)) \wedge Finishes(t, pre1(e)) \wedge$$
$$Working(c, eff1(e)) \wedge Meets(t, eff1(e)).$$

With this information, the agent knows what it means for the car to be fixed (although the agent does not know how it was done). Furthermore, such axioms could be used by a system to recognize situations in which the car has been fixed. Knowledge of this sort is also essential for much of natural language semantics, where many verbs are defined

and used without the agent's knowing the necessary causal knowledge. Allen (1984) discusses this at length.

To make this distinction concrete, we can partition our axioms describing events and actions into three broad categories:

EDEF Event definitions – These are axioms of the form

$$\forall e \,.\, E(e) \wedge \phi \supset \psi,$$

where E is an event-type predicate and ϕ and ψ contain no event predicates. The necessary conditions for stacking given in the previous section fall into this category.

ETRY Action definitions – These are axioms of the form

$$\forall t, \ldots \,.\, Try(\pi, t) \wedge \phi \supset \exists e \,.\, E(e) \wedge t \circ time(e) \wedge \psi,$$

where again E is an event predicate. In this case, ψ represents constraints on e, and can involve other quantified variables. The symbol "$\circ$" stands for the temporal relation between the action and the event, typically "$Equal$" or "$Meets$", as discussed above.

EGEN Event generation axioms – These are of the form

$$\forall e \,.\, E(e) \wedge \phi \supset \exists e' \,.\, E'(e') \wedge \psi.$$

Again, E and E' are event-type predicates and ϕ and ψ represent constraints on the events. The classic example of event generation is represented in English using the "by" locution, for example, signaling a turn by waving one's arm, under appropriate conditions. More concrete examples will be presented in later sections.

Examples of all three classes of axioms will be presented in the next section when we describe application of the formalism to a set of problems from the literature on reasoning about action.

2.2.5 Discussion and Related Work

Having introduced our representation of time, actions and events, we can now compare it to the other major formalisms. Note that this comparison is based on the expressiveness and simplicity of the logic for capturing commonsense knowledge. Dealing with prediction is a separate issue for most of these formalisms, and will be discussed at the end of the next section.

As discussed previously, most formalisms have not included an explicit model of time, but base their model on states or situations. To handle explicit temporal relations in the situation calculus, a function can be defined that maps a state or situation to a timepoint. In this way, the situation calculus defines a point-based branching time model. A duration function on actions can be defined that allows one to compute the

time of the resulting situation given the initial situation (Gelfond et al., 1991). Within the constructive models, however, the range of temporal reasoning that can be performed is severely limited. First, note that there are no situations defined during the time an action is executing. This means that you cannot assert anything about the world during an action execution. Rather, all effects of an action must be packaged into the resulting situation. As a consequence, all effects start simultaneously. Since situations correspond to time points, the formalism also has no mechanism for representing knowledge about how long an effect might hold, as time only moves forward as a result of an action. The only way around this seems to be to allow null actions that move time forward but have no effect, which is neither convenient or intuitive. In the temporal logic with events, you have the full power of the formalism for representing information about the temporal properties of events and their effects. In addition, effects P and Q of some event are each defined to hold over separate intervals. These intervals are independent of each other, and so we can reason about how long P remains true without worrying about how long Q holds.

As a consequence, the only worlds easily represented by the constructive situation calculus are those that remain static except when the agent acts, and where nothing important happens while actions are being executed. Of course, if one abandons the requirement of a constructive theory, then more complicated scenarios can be represented. In fact, in some approaches (*e.g.*, (Shanahan, 1990)) even continuous change has been represented. The primary mechanism that enables this, however, is the exploitation of the mapping from situations to times, and the introduction of situations that are not the result of any actions. Given these radical extensions, it is not clear to us what the advantage is in retaining the situation calculus as the underlying theory, rather than moving to the simpler, more direct, explicit temporal logic. Temporal logic gives properties, actions and events equal temporal status, allowing complex situations to be captured quite directly in a manner similar to how they are intuitively described.

We have also made a distinction between actions and events that is not made in most other formalisms, which use only one construct. So an interesting question is whether these other representations involve actions or events? This question is hard to answer for STRIPS-based systems, as they say nothing about unsuccessful attempts of actions. A STRIPS-style system, for instance, only gives information about the effect of an action when the preconditions holds (*i.e.*, when the action is guaranteed to succeed). There need be no distinction made between actions and events in these circumstances, as there is a one-to-one mapping between actions and the events that they cause.

The situation calculus, on the other hand, does allow multiple axioms defining the consequences of an action, so the same action does not always have the same effect. This seems close to our intuitions about actions. For instance, the situation calculus allows the following axioms, which state how the stacking action might result in different situations when performed in different circumstances:

> If a and b are clear in situation s, then $On(a, b)$ is true in situation $Result(stack(a, b), s)$.
> If a is clear, and $On(c, b)$ is true in situation s, then $On(a, c)$ is true in situation $Result(stack(a, b), s)$.

There is no explicit representation of events in the situation calculus, which makes it difficult to represent knowledge of the actual world. For instance, it is not possible to directly assert that an event E will occur tomorrow, or even that the agent will perform an action A tomorrow. In the constructive situation calculus, one would have to quantify over all action sequences and ensure that the event or action was present. This does not seem workable, and recent attempts to represent the actual events generally abandon the constructive approach, and have to introduce an explicit new construct for events (*e.g.*, (Miller and Shanahan, 1994)). The resulting formalism is again closer to what we propose, and we question the advantage of retaining the original situation calculus framework.

The event calculus (Kowalski and Sergot, 1986) has an explicit notion of event that corresponds more closely to our notion of events. This is not surprising as both this formalism and our own development of the temporal logic are based on intuitions about the close relationship between time and events. The event calculus does not seem to have a separate notion of actions, although formulas of the form $act(E, give(a, x, y))$ are used that suggest they might. As far as we can tell, however, this is only a syntactic mechanism for naming events. Reasoning in the event calculus is driven by the events that occur, as it is in our approach, but the relationship between the behaviors that an agent might perform and the events that the agent causes is not explored. Their representation of events seems weaker than what we are proposing. Specifically, the formalism only allows the simplest temporal relationships between an event and its effects (in our terms, all effects are "$MetBy$" the event), and it is not clear how events could be extended in time. The most significant difference, however, is that the event calculus formalism is organized around a specific strategy for dealing with the frame problem. Specifically, it uses a negation as failure to implement a persistence strategy where a proposition remains true until an event explicitly changes (or clips) it. For this strategy to be valid, assumptions must be made that are similar to the STRIPS assumptions.

Closest to our representation is that described by McDermott (1985). He does distinguish between actions (which he calls tasks) and events, and uses an explicit model of time. As a result, he can define notions such as a successful action attempt, and explicitly reason about an agent's actions and what events they will cause. His representation of time has many similarities to the work that extends the situation calculus with a time line (Pinto, 1994), and he explores the consequences of adding explicit time to such a model in depth. A significant difference between our approaches is that we use an interval-based logic, while McDermott uses a continuous point-based model. These differences have been discussed in detail elsewhere (*e.g.*, (van Benthem, 1983; Allen, 1984; Shoham, 1988)).

2.3 Reasoning about Action in Simple Domains

This section discusses prediction in theories of action, in particular the frame problem, and describes our approach based on explanation closure. The approach is illustrated by formalizing and solving a set of simple problems from the literature on

reasoning about action. Extensions of the theory to more complicated and realistic domains are presented in subsequent sections.

2.3.1 Prediction

As mentioned in the introduction, there are three major reasoning tasks we require the logic of events and actions to support: prediction, planning, and explanation. Of these, prediction is the most basic capability—given a description of the world and a set of actions and events that will occur, what can we conclude about the past, present, and future? With a temporal logic, the initial world description might already contain some information about the past or the future, say some external events that will occur, or future actions that the agent knows it will perform. The prediction task reduces to predicting the effects of new actions and events that are posited to occur, and updating the world model accordingly. Neither planning nor explanation can be accomplished without a model of prediction. In planning, for instance, the task is to find a set of actions that will accomplish a given set of goals. This can be divided into two abstract tasks: generating a candidate set of actions, and evaluating whether the plan will succeed. This latter task is exactly the prediction task. Explanation can be similarly decomposed into generating a possible set of events that might explain the observations, and then verifying whether the events would actually cause the observed effects. Of course, any particular algorithm might not divide the reasoning into two explicit steps. In fact, most planning algorithms exploit a specific prediction model in order to suggest actions likely to produce a good plan, but all systems are based on some prediction model.

As a concrete example, consider the standard backward chaining planning algorithm using the STRIPS representation (*e.g.*, (Nilsson, 1980)). The algorithm chains backwards from the goal state. First, the goal state is compared to the initial state and a set of propositions that differ in truth value between the two states are found. Then, one of these propositions is chosen and an action is introduced that has the proposition as one of its effects. Then the state of the world before this action is performed is computed using a technique called regression, which essentially inverts the add and delete lists in the action's definition. This new state now becomes the goal state and the process continues until the initial state is derived. Once that is done, the algorithm has found a sequence of actions leading from the initial state to the goal state as desired. The prediction model that makes this approach valid includes the following assumptions:

- No other events or changes occur in the world except for the planned actions;
- The action definitions completely describe all changes that occur as the result of the action.

With these two assumptions, prediction is accomplished simply by applying the transformations defined by each of the actions. The regression technique was designed so that an explicit prediction step was not necessary, given the assumptions. Specifically, in regressing an operator back from a state s to a preceding state s', one is guaranteed that predicting from s' with the same action would yield back s. By exploiting

this property, the plan can be constructed in this backward fashion, and once a plan is found, it is guaranteed to achieve the goal given the two assumptions above.

The same prediction model underlies the formalisms based on non-linear planning as in TWEAK (Chapman, 1987) and SNLP (McAllester and Rosenblitt, 1991) and systems based on these techniques. The additional complication is that actions are partially ordered, and so systems must distinguish between predictions that are necessarily true (in any allowable action ordering) or only possibly true (in some allowable action orderings). With this addition, the new prediction model validates the proposed search strategies. Similar methods using the event calculus are proposed in (Eshghi, 1988; Shanahan, 1989; Kowalski, 1992).

2.3.2 Frame Axioms and Explanation Closure

The problem with the above approach is that the STRIPS assumptions do not hold in most realistic situations that one needs to reason about. The situation calculus and temporal logic-based approaches do not have to operate with such assumptions. As a result, these theories themselves make little commitment to how the state resulting from an action relates to the state before the action. Rather the properties of the resulting state must be specified axiomatically, and the frame problem involves how best to specify these properties. The original proposal for the situation calculus (McCarthy and Hayes, 1969) was to use *frame axioms*, which explicitly stated which properties are not changed by the actions. Explicit frame axioms have come under criticism, primarily because there are too many of them, both to write down explicitly and to reason with efficiently.

There are several ways to try to overcome this problem. Many researchers abandon frame axioms altogether, and have built models that use persistence or inertia assumptions (*e.g.*, (Lifschitz, 1987; Shoham, 1988)). These approaches assume that all changes caused by an action are specified, and every property not asserted to change does not change. This technique has much to recommend it, as it eliminates the need to enumerate frame axioms, but in its simple form it tends to be too strong. In particular, if there is uncertainty as to whether a property might change or not, techniques based on this approach will often incorrectly assume that the change does not occur. Other approaches work instead by minimizing event or action occurrences. Properties are assumed to change only as a result of events defined in the representation, and logically unnecessary events do not occur (*e.g.*, (Georgeff, 1986b,a; Morgenstern and Stein, 1988)). These approaches show more promise at handling more complex situations and have many similarities to our work.

The approach we take, however, retains the flavor of explicit frame axioms. Rather than specifying for each action whether each property changes or not, however, one specifies for each property what events can change it. The problem of reasoning about changes then reduces to the problem of reasoning about what events may or may not have occurred. This technique is called the *explanation closure* approach and was proposed by Haas (1987) and Schubert (1990). Schubert has shown that such a technique can dramatically reduce the number of frame axioms required to produce a workable

set of axioms for a problem. Of course, assumptions must still be made. As in other approaches, we assume that unnecessary events do not occur, but this is specified axiomatically rather than being built into the semantic model. This has several advantages. Most importantly, the resulting axioms can be interpreted with the standard semantics of the first-order predicate calculus, meaning that there are well-defined notions of entailment and proof. We can show that our representation handles a particular class of examples by showing a proof of the desired consequences, without needing to appeal to model-theoretic arguments in a non-standard semantics. In addition, various forms of uncertainty are handled using the standard methods in logic with disjunction and existential quantification. Finally, the assumptions that are made appear explicitly as axioms in the system. As noted at the outset, this important point will be used in the development of our plan representation in the next chapter.

If you are willing to reinstate strong assumptions about the domain, roughly equivalent to the STRIPS assumptions, then Reiter (1992) has shown that the explanation closure axioms can be computed automatically using predicate completion techniques. But this close correspondence breaks down in more complex domains. Schubert (1990) argues that explanation closure is distinct from predicate completion or biconditionalization, and thus that the axioms cannot be generated automatically. Specifically, he argues that any automatic procedure will produce axioms that are too strong in cases where knowledge of the world is incomplete and actions may have conditional effects. In addition, he argues that explanation closure axioms have "epistemic content" and are independently motivated by other problems such as language understanding. As such, they form a crucial body of knowledge necessary for many commonsense reasoning tasks. Notwithstanding this issue, it seems likely that explanation closure axioms can be generated automatically in many cases, for instance for actions that never have conditional effects. But this is a programming issue rather than a logical issue. As far as the formalism is concerned, the closure axioms are an essential part of the reasoner's knowledge of the domain are are not automatically derivable. As part of a practical knowledge representation, all sorts of techniques could be developed to make the specification of this knowledge easier, but these concerns are not the subject of this investigation.

The examples to follow in the rest of this section do not demonstrate the full power of our approach, as they are formulated not to involve simultaneous actions and external events. But they will facilitate the comparison of our work to other approaches, and the more complex issues will be considered in later sections. Because of the simple nature of the problems, the closure axioms can be classified into two classes, corresponding to two assumptions: no properties change unless explicitly changed by an event occurring, and no events occur except as the result of the actions. Although these are difficult to precisely describe schematically, they look something like the following following:

EXCP (Strong Closure on Properties) Every property change results from the occurrence of an instance of one of the event types defined in the axioms. These axioms are of the form:

$$\forall t, t' \,.\, P(t) \wedge \neg P(t') \wedge t : t' \supset \exists e \,.\, (E_1(e) \vee E_2(e) \vee \ldots) \wedge \mathit{time}(e) : t',$$

where the E_i are the event-type predicates that (possibly) affect the truth value of P. These axioms are derived from the event definition axioms (EDEF).

EXCE (Strong Closure on Events) Every event that occurs does so as a result of some action being attempted, possibly indirectly via event generation. These axioms are of the form: "$\forall e \,.\, E(e) \supset \phi_1 \vee \phi_2 \vee \ldots$," where ϕ_i is either of the form

$$\exists t \,.\, Try(\pi, t) \wedge t \circ time(e) \wedge \psi,$$

for π an action term and "$\circ$" a temporal relation (typically "$=$"), or of the form

$$\exists e' \,.\, E'(e') \wedge time(e) \circ time(e') \wedge \psi,$$

for E' an event-type predicate. These axioms can often be derived from the action definition (ETRY) and event generation (EGEN) axioms.

2.3.3 The Sandewall Test Suite for Reasoning about Action and Change

By way of illustrating the application of the logic and the explanation closure technique, we now present formalizations of some of the standard problems from the literature on reasoning about action gathered in (Sandewall, 1994). These are mostly variants of the Yale Shooting Problem (Hanks and McDermott, 1986), a basic AI test case for reasoning about action. Each scenario involves a very simple domain where a single problem is presented and thus it is not the best vehicle for demonstrating the generality of our approach. However, it does establish a certain baseline and allows us to support our claims as to the expressiveness and naturalness of the event-based temporal logic.

To begin, consider our formulation of the basic Yale Shooting Problem (YSP). The scenario involves a hapless victim, Fred, who apparently sits idly by while an adversary loads a gun, waits some period of time, and shoots (at Fred). The question is whether Fred is dead or not, or rather whether an axiomatization of the problem in some logical system allows the conclusion that Fred is dead to be derived. This simple problem has generated an extensive body of literature (*cf.* (Brown, 1987)). In most cases, the issue is that while the loadedness of the gun ought to persist through the waiting (and then the shooting succeeds in killing Fred), Fred's "aliveness" ought not to persist through the shooting (indeed, cannot, without risk of inconsistency) although it should persist through the loading and waiting. On the other hand, there might be "non-standard" models where the gun somehow becomes unloaded, in which case the shooting would fail to kill Fred.

Figure 2.7 shows the axioms for the YSP in our logic; they are depicted graphically in Figure 2.8. Recall that the EDEF axioms describe the necessary conditions for event occurrence and the ETRY axioms describe the sufficient conditions relating program executions (actions) and events. Of course, the example axioms are very simple—there are only temporal roles and there are no generation axioms (yet).

Figure 2.9 gives the explanation closure axioms corresponding to the event and action axioms in Figure 2.7. Given the simple actions and strong assumptions made in the

EDEF1 $\forall e \,.\, \textsc{Load}(e) \supset \mathit{Loaded}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e)$

EDEF2 $\forall e \,.\, \textsc{Shoot}(e) \supset \mathit{Loaded}(\mathit{pre1}(e)) \wedge \mathit{SameEnd}(\mathit{pre1}(e), \mathit{time}(e)) \wedge \neg\mathit{Loaded}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e) \wedge \neg\mathit{Alive}(\mathit{eff2}(e)) \wedge \mathit{time}(e) : \mathit{eff2}(e)$

ETRY1 $\forall t \,.\, \mathit{Try}(\mathit{load}, t) \supset \exists e \,.\, \textsc{Load}(e, t)$

ETRY2 $\forall t \,.\, \mathit{Try}(\mathit{shoot}, t) \wedge \mathit{Loaded}(t) \supset \exists e \,.\, \textsc{Shoot}(e, t)$

Figure 2.7: Basic axioms for the Yale Shooting Problem

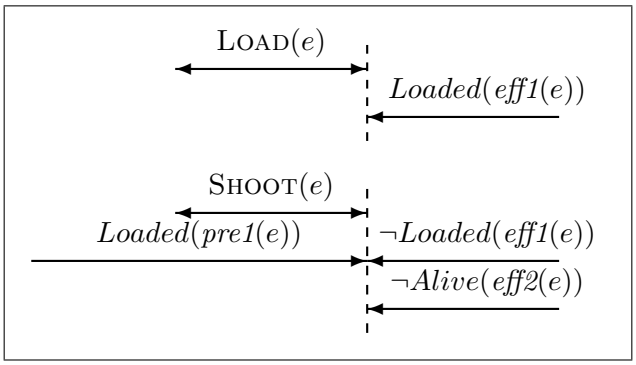


Figure 2.8: The LOAD and SHOOT event definitions

EXCP1 $\forall t, t' \,.\, \neg Loaded(t) \wedge Loaded(t') \wedge t : t' \supset \exists e \,.\, \textsc{Load}(e) \wedge time(e) : t'$

EXCP2a $\forall t, t' \,.\, Loaded(t) \wedge \neg Loaded(t') \wedge t : t' \supset \exists e \,.\, \textsc{Shoot}(e) \wedge time(e) : t'$

EXCP2b $\forall t, t' \,.\, Alive(t) \wedge \neg Alive(t') \wedge t : t' \supset \exists e \,.\, \textsc{Shoot}(e) \wedge time(e) : t'$

EXCE1 $\forall e \,.\, \textsc{Load}(e) \supset \exists t \,.\, Try(load, t) \wedge t = time(e)$

EXCE2 $\forall e \,.\, \textsc{Shoot}(e) \supset \exists t \,.\, Try(shoot, t) \wedge t = time(e)$

Figure 2.9: Explanation closure axioms for the YTS

problem formulation, an automatic procedure could probably devised to generate them. But notice that the closure axioms are not simply a strengthening of the implication to a biconditional. In particular, the EXCP2b axiom does not say that every SHOOT event causes a transition from alive to dead, just that every such transition is the result of a SHOOT. This would allow one to shoot things that are already dead, for example.

In the remainder of this section, we use these axioms for several of the core problems of the Sandewall test suite (Sandewall, 1994). Each of these problems deals with prediction from a given situation, sometimes predicting previous facts (retrodicting). For each problem, we provide an axiomatization of the problem description and a proof (or proof sketch) of the conclusion. The role of Fred is taken by a turkey in the Sandewall collection, hence the change in terminology.

Yale Turkey Shoot (YTS)

This is the original YSP scenario described above. Initially the turkey is alive and the gun is not loaded. The agent loads the gun, waits, and shoots. We are to show that the turkey is dead sometime after the shooting. At issue is the persistence of loadedness through the waiting and the non-persistence of aliveness through the shooting. The following axioms describe the scenario:

AX0 $t0 \prec t1 \prec t2 \prec t3$

AX1 $Alive(t0)$

AX2 $\neg Loaded(t0)$

AX3 $Try(load, t1)$

AX4 $Try(shoot, t3)$

AX5 $\forall a, t \,.\, Try(a, t) \equiv (a = load \wedge t = t1) \vee (a = shoot \wedge t = t3)$

The final axiom is the important assumption that we attempt only the given actions. With the EXCE axioms, this ensures that "nothing else happens," which drives the explanation closure method. Note that the formalism does not require such an assumption. Rather, we are making explicit an aspect of the scenario that is usually implicit and accomplished via some form of semantic minimization.

To Prove: $\exists t \,.\, \neg \mathit{Alive}(t) \wedge t3 \prec\!: t$

Proof: Since there are no preconditions for loading, AX3 and ETRY1 give

$$\exists e_1 \,.\, \textsc{Load}(e_1, t1),$$

then EDEF1 gives

$$\mathit{Loaded}(\mathit{eff1}(e_1)) \wedge t1 : \mathit{eff1}(e_1).$$

That is, the loading succeeds, and the gun is loaded, at least immediately afterwards. Now, suppose that

$$\mathit{Loaded}(t3) \qquad\qquad (*)$$

that is, the gun remains loaded until the shooting is attempted. In that case, AX4 and ETRY2 give $\exists e_3 \,.\, \textsc{Shoot}(e_3, t3)$, and then EDEF2 gives

$$\neg \mathit{Alive}(\mathit{eff2}(e_3)) \wedge t3 : \mathit{eff2}(e_3).$$

That is, the shooting also succeeds and the turkey is dead afterwards, as required.

To show the persistence $(*)$, suppose otherwise that $\neg \mathit{Loaded}(t3)$. The interval temporal logic theorem DISJ (Section 2.2.2) gives $\mathit{eff1}(e_1) \bowtie t3$, then AX0 and the fact that $t1 : \mathit{eff1}(e_1)$ give $\mathit{eff1}(e_1) \prec\!: t3$. We apply interval temporal logic theorem TRPT to get

$$\exists T, T' \,.\, \mathit{Loaded}(T) \wedge \neg \mathit{Loaded}(T') \wedge T : T' \wedge t3 \sqsubseteq T'.$$

Then, EXCP2a gives

$$\exists e \,.\, \textsc{Shoot}(e) \wedge \mathit{time}(e) : T'.$$

That is, if the gun became unloaded it must have been because of a shooting. Note that $\mathit{time}(e) \prec\!: t3$ since $t3 \sqsubseteq T'$ and $\mathit{time}(e) : T'$. Then from EXCE2 we get

$$\exists t \,.\, \mathit{Try}(\mathit{shoot}, t) \wedge t = \mathit{time}(e),$$

i.e., if there was a shooting then someone must have tried to shoot. Since $t = \mathit{time}(e) \prec\!: t3$, we have $t \neq t3$, contradicting AX5. □

Note that this proof will go through in exactly the same way even if you add other actions, such as waiting or eating lunch, at time $t2$, as long as the definition of such actions and their closure axioms indicate that they do not effect the gun being loaded. For example, the action of waiting for time to pass would be simply axiomatized as follows, assuming there were no preconditions for waiting:

ETRY3 $\forall t \,.\, \mathit{Try}(\mathit{wait}, t) \supset \exists e \,.\, \textsc{Wait}(e, t).$

Since the event has no effects, there is no need for an EDEF axiom and no need to change the EXCP axioms. There would, of course, be a corresponding EXCE axiom stating that waiting events only occur if the agent tries to wait. It should then be clear that adding $\mathit{Try}(\mathit{wait}, t2)$ to the axiomatization of the YTS scenario and adjusting AX5 accordingly will not affect the proof that the turkey is dead. Adding a more complicated action,

such as eating lunch, with preconditions and effects would not be any different, provided it didn't affect the validity of the closure axioms (as it shouldn't if it is independent of loading and shooting).

As Schubert (1992) has remarked, the formulation based on explanation closure makes strong assumptions about the situation, as reflected in the closure axioms. In particular, the axiom AX5 that "nothing else happens" rules out many of the "problems" that arise in nonmonotonic approaches to this problem, such as ghosts appearing and unloading the gun during the interval between loading and shooting. Is this reasonable? It certainly seems more reasonable to make such an assumption explicit (as with AX5) and use a standard, well-understood logic than to build the assumptions into the semantics of the logic to make the intuitive solutions fall out as theorems. Such an assumption might be justified by the agent's knowledge of its plans, or by conventions about how stories are generally told, for example.

Stanford Murder Mystery (SMM)

In this variant, the turkey is initially alive but nothing is known about the state of the gun. The agent shoots and the turkey is observed to be dead after the shooting. We are to conclude that the gun was initially loaded. That is, we are required to "predict" aspects of the situation backwards in time (sometimes called retrodiction). This can be difficult for reasoners that "move" forward in time.

We have the following axioms:

AX0 $t0 \prec t1 \prec t2$

AX1 $Alive(t0)$

AX2 $\neg Alive(t2)$

AX3 $Try(shoot, t1)$

AX4 $\forall a, t \,.\, Try(a, t) \equiv (a = shoot \wedge t = t1)$

To Prove: $Loaded(t0)$

Proof: To start, AX0, AX1, AX2, and TRPT give

$$\exists T, T' \,.\, Alive(T) \wedge \neg Alive(T') \wedge T : T' \wedge t2 \sqsubseteq T',$$

then EXCP2b gives

$$\exists e \,.\, \textsc{Shoot}(e) \wedge time(e) : T'.$$

That is, a shooting event must have occurred between $t0$ and $t2$ to account for the death of the turkey. Then EDEF2 gives

$$Loaded(pre1(e)) \wedge SameEnd(pre1(e), time(e)),$$

i.e., since the shooting was successful, the gun must have been loaded.

Now, suppose $\neg Loaded(t0)$. Then DISJ, AX0, and the temporal constraints on $pre1(e)$ give $t0 \prec: pre1(e)$, then TRPT and EXCP1 give

$$\exists e' \,.\, \textsc{Load}(e') \wedge time(e') : T'',$$

where $pre1(e) \sqsubseteq T''$. That is, there must have been a loading event if the gun was initially unloaded and subsequently loaded. Applying EXCE1 gives

$$\exists t' \,.\, Try(load, t') \wedge t' = time(e'),$$

which contradicts AX4 as we know there was no loading attempted. Thus the gun must have initially been loaded, *i.e.*, $Loaded(t0)$. □

We see that the temporal logic is immune to the "direction" of the proof with respect to the direction of time, as expected. In contrast, most other models allow temporal prediction only in a forward direction (*e.g.*, (Dean and McDermott, 1987)) and must use a completely different approach for "persisting" backwards in time.

Interestingly, the proof for the SMM does not use the fact that we tried to shoot (AX3). Rather the facts that (a) the turkey died (AX1, AX2), (b) the only way it could have died was if we shot it (EXCP2b), and (c) we didn't try to do anything but shoot (AX4, in particular, we didn't load), were sufficient to prove the goal. The fact that we tried to shoot turns out to be a consequence of these.

Russian Turkey Shoot (RTS)

This variant introduces a new action, that of spinning the barrel of the gun, that randomizes its loadedness. At issue is the ability of the representation to deal with uncertainty in the effects of actions. The following axiom describes spinning the barrel:

ETRY3 $\forall t \,.\, Try(spin, t) \supset \exists e \,.\, \textsc{Spin}(e, t)$

There is no EDEF axiom for *Spin*, since the effect of spinning is the tautology $Loaded \vee \neg Loaded$. However, the explanation closure axioms (Figure 2.9) need to be modified to accommodate spinning:

EXCP1 $\forall t, t' \,.\, \neg Loaded(t) \wedge Loaded(t') \wedge t : t' \supset$
$\exists e \,.\, (\textsc{Load}(e) \vee \textsc{Spin}(e)) \wedge time(e) : t'$

EXCP2a $\forall t, t' \,.\, Loaded(t) \wedge \neg Loaded(t') \wedge t : t' \supset$
$\exists e \,.\, (\textsc{Shoot}(e) \vee \textsc{Spin}(e)) \wedge time(e) : t'$

EXCE3 $\forall e \,.\, \textsc{Spin}(e) \supset \exists t \,.\, Try(spin, t) \wedge t = time(e)$

In this case the explanation closure axioms include information not present in the original axioms, namely that spinning is a possible cause of the gun becoming unloaded, despite the lack of an EDEF axiom stating that it necessarily causes the gun to become unloaded.

The problem setup is the same as for YTS, but the agent loads, spins, and then shoots. We should now no longer conclude that the turkey dies. The axioms are the same as for the YTS with the addition of AXs that says the agent spins, and the modification of AX5 to allow spinning.

AX0 $t0 \prec t1 \prec t2 \prec t3$

AX1 $Alive(t0)$

AX2 $\neg Loaded(t0)$

AX3 $Try(load, t1)$

AXs $Try(spin, t2)$

AX4 $Try(shoot, t3)$

AX5 $\forall a, t \,.\, Try(a, t) \equiv$
$(a = load \wedge t = t1) \vee (a = spin \wedge t = t2) \vee (a = shoot \wedge t = t3)$

Proof: With these changes, the proof starts as for YTS, but in attempting to refute the hypothesis that the gun remains loaded ($*$), we now get

$$\exists e \,.\, (\textsc{Shoot}(e) \vee \textsc{Spin}(e)) \wedge time(e) : T'$$

for $t3 \sqsubseteq T'$ from the modified EXCP2a. Then EXCE2 and EXCE3 give

$$\exists t \,.\, (Try(shoot, t) \vee Try(spin, t)) \wedge t = time(e),$$

which no longer contradicts the modified AX5. □

We see that the temporal logic with explanation closure can accommodate uncertainty in the effects of actions, so long as their potential occurrence is taken account of. Of course, the conclusions are weaker, but that is a consequence of the uncertainty inherent in the problem, not a shortcoming of the logic.

Dead Xor Alive (DXA)

For this variant, we replace $\neg Alive$ by $Dead$ as the effect of successful shooting, and add an axiom

$$\forall t \,.\, Dead(t) \equiv \neg Alive(t).$$

The logical equivalence leads to what Sandewall terms "autoramifications," but poses no problem for the deductive approach presented here.

Walking Turkey Problem (WTP)

Similarly to DXA, we are given that the turkey is known to be walking initially, but not explicitly known to be alive. We have the axiom

$$\forall t \,.\, Walking(t) \supset Alive(t),$$

however, and we are to conclude that the turkey is both dead and not walking after shooting. The proof of death is exactly like YTS after one application of the new axiom, and then the contrapositive of that axiom is used to show that the turkey is not walking once dead.

2.3.4 Discussion and Related Work

In all the examples presented above, the central issue is how to reason formally about what changes and what doesn't change as the result of some action, *i.e.*, the frame problem. Work on the frame problem investigates methods of making assumptions about the world to predict the likely consequences of actions. There are therefore two separable issues to consider, which have been called the "epistemological" and "computational" aspects of the problem (*cf.* (Kowalski, 1992), but also (McCarthy and Hayes, 1969) regarding epistemological and heuristic adequacy). The epistemological aspect concerns what assumptions one makes about the world, while the computational aspect concerns how to compute and use these assumptions in the formalism. For example, it is an epistemological issue whether to make assumptions about property changes as in the persistence-based approaches, or to make assumptions about event occurrences. On the other hand, it is a computational issue whether to use minimization techniques in the model theory to implement assumptions, or to use explicit axioms, as in the original situation calculus or with the explanation closure technique. We will discuss each of these issues separately, although they are, of course, closely related.

Our approach is based on making assumptions about event occurrence, both events caused by actions and external events. In this way, it is similar to work by Morgenstern and Stein (1988), Haas (1987), Schubert (1990), and Kowlaski and Sergot (1986). In such approaches, properties do not change unless there is some event that causes them to change. Approaches based on STRIPS-style representations can be viewed in this way as well, except that the assumptions are not a separable part of the formalism. Rather, the representation only makes sense when these assumptions are made. The other major approach focuses on minimizing property change, with additional constraints based on the temporal ordering of properties (*e.g.*, (Shoham, 1988; Kautz, 1986)) or minimizing causal relationships (*e.g.*, (Lifschitz, 1987)). Sandewall (1994) examines the advantages and limitations of each of these approaches in detail. Most of the approaches that minimize property change cannot handle Sandewall's test suite of problems, let alone be extended to handle external events and simultaneous actions. We find that basing the assumptions on events leads to much more intuitive characterization of problems, where each statement in the logic is closely related to an intuitive fact about the world. In addition, as we show in the subsequent sections, this approach naturally handles a wide range of more complex problems.

The second issue is what mechanism is used to make the assumptions. There are two main approaches here: explicitly adding axioms that encode the assumptions (*e.g.*, (Green, 1969; Schubert, 1990)) or using a nonmonotonic model theory that defines a new notion of entailment that includes the assumptions (*e.g.*, (McCarthy, 1980; Shoham, 1988; Baker, 1991; Sandewall, 1994)). Of course, the work on circumscription shows that model-theoretic techniques always have an equivalent axiomatic formulation, although it may require going beyond standard first-order logic. This equivalence suggests that there is really a continuum of approaches here. Everyone must make assumptions. Some prefer to pack it all in the model theory, others use axioms to capture much of the complexity and use only simple minimization assumptions, while others prefer to encode everything in axioms. Many of the issues come down to ease of formalization,

an issue that is bound to vary from researcher to researcher. The reason we prefer explanation closure axioms is that they give us a very flexible system that is easily extended to handle complex issues in representing actions. In addition, the resulting representation is a standard first-order logic, so it is relatively straightforward to tell if certain consequences follow from the axioms. The same cannot be said for approaches based on specialized reasoners or specialized semantic theories that solve one problem at the expense of the others. Good evidence for this claim is seen in the fact that the community has seen the "Dead-Xor-Alive" problem and the Walking Turkey problem as problems at all. They should be trivial consequences of any reasonable theory of action.

Explanation closure axioms allow us to treat each event class on a case-by-case basis. This allows idiosyncratic events and properties to be represented that don't follow the norm. It is much harder to have such flexibility in the model minimization approaches, as all events and properties tend to be treated uniformly. To handle exceptional cases usually requires extending the syntax of the language to introduce special classes of predicates that are minimized in different ways, leading to more complex and unintuitive formulations.

Some common criticisms of explanation closure approaches are that it is too difficult to write down all the axioms, and that the approach is cheating because we have encoded the solution in the axioms. Schubert (1990) provides a good response to the first criticism, namely that the explanation closure axioms are an essential part of an agent's commonsense knowledge of the world, and are needed independently of the frame problem in tasks such as plan recognition or natural language understanding. In addition, just because the theory is based on these axioms doesn't mean that a programmer would have to write them all down. In some cases, the axioms can be automatically computed (Reiter, 1992) or generated on the fly. In an actual application, we can take advantage of such methods whenever possible. As for the issue of cheating, it doesn't seem to us that explicitly encoding the assumptions that make the representation work is any more cheating than hiding the assumptions in a complex model theory. We must remember that these really are assumptions, and they may be wrong. If we are ever to reason about cases where assumptions have been made and shown to be false, they need to be explicit in the representation.

2.4 External Events

External events arise for different reasons. Some simply occur as a result of natural forces in the world, whereas others are set into motion by the action of the planning agent. Some external events are predictable—we know, for instance, that the sun will rise tomorrow, and that many of the central events of a normal workday will occur. But even when we are certain that a particular event will occur, we are often still uncertain as to when it will occur. Most of the time, for instance, I can only roughly estimate when the sun will rise, but this uncertainty doesn't generally affect my plans. Rather, I know that by a certain time, say 6AM at this time of year, the sun will be up. In most realistic applications, the planning agent will not have complete knowledge of the

world and its causal structure, and thus there will be many external events for which it cannot reliably predict whether they will occur or not. Assuming that such events do not occur will lead to highly optimistic plans. Rather, an agent should be able to plan given the uncertainty of external events.

For now, we will treat actions by other agents as external events. That is, there is no difference between events that occur because of natural forces and events that occur because of other agent's actions. In a full theory, the main difference between the two classes would result from additional knowledge that is available when reasoning about other agents. In particular, knowledge that other agents are rational will sanction plan recognition inferences, for example.

The logic presented in Section 2.2 already provides the formalism for representing external events. In fact, the only way to distinguish external events from events caused by the agent's acting is by the presence of an axiom involving the predicate $Try(\pi, t)$, that states that the event was caused by an action attempt. But there is no requirement that all events have axioms relating them to action attempts.

The complications arise in characterizing the appropriate explanation closure axioms. In particular, if one makes the strong event closure assumptions (as in the previous section), that all events are ultimately caused by some action, then external events caused solely by natural forces or other agents cannot be represented. Of course, the explanation closure approach doesn't require such a strong assumption. If fact, we could simply not have closure axioms for some event classes that are caused by natural forces. But this would then allow such events to interfere with the prediction process at every possible opportunity, as you could never prove that the event didn't occur. While this might be the right result in some situations, usually external events are more constrained.

There are two orthogonal aspects of external events that affect the representation. The first relates to uncertainty about whether the event will occur. A non-probabilistic logic can only easily represent the two extremes on this scale: either the event definitely will occur, or it may or may not occur. The second relates to the conditions for an event's occurrence. Again, there is a continuum here between a complete lack of constraints on when the event may occur, and cases where the event will only occur under specific conditions, such as being caused by the agent's actions. Somewhere in the middle of this scale would be events that may only occur within a certain time period, but that are independent of the agent's actions. In this section, we will consider three common combinations that appear frequently in many domains. These are:

- Triggered events: The external event will not occur unless it is specifically triggered by an agent's actions (*e.g.*, the microwave will heat my coffee only if someone presses the "on" button).
- Definite events: The external event will definitely occur, independent of the agent's actions, although the exact time may be uncertain (*e.g.*, the sun will rise between 5:30 and 6:30AM).
- Spontaneous events: The external event may or may not occur during some time period (*e.g.*, I might win the lottery tonight).

Our representation can also handle more complex cases, such as triggered spontaneous events, which only become possible as a result of some action by the agent.

2.4.1 Triggered Events

Triggered events are actually already handled by the development so far. In fact, all events in the last section can be viewed as triggered events that characterize the agent's actions. The same approach can be used to handle other triggered external events.

Sandewall's Hiding Turkey Scenario is quite naturally handled as a simple case of reasoning about triggered external events. In this setting, the turkey may or not be deaf. If it is not deaf, then it will go into hiding when the gun is loaded, and thus escape death from the shooting. That is, we are given information about how the turkey will respond to a certain situation, namely that it will hide if it hears the gun being loaded. We must conclude that after the shooting, either the turkey is alive and not deaf, or it is dead.

Many formalizations of this problem treat the turkey hiding as an conditional effect of loading the gun rather than as an external event. If this approach worked for all triggered external events, then it might be justified. But unfortunately, it only works in the simplest of cases, namely when the event caused has no structure or temporal complexity, and so can be characterized as a static effect. For example, consider a situation where the only place the turkey might hide is in a box, and the box has a door that can be closed by the agent. A natural formalization of this would include the following facts:

- If the turkey is not deaf, it will try to hide when the gun is loaded.
- If the box door is open when the turkey tries to hide, it will be hidden in 30 seconds.

You could capture some of this situation by adding a new effect rule about loading that if the box door is open, the turkey will hide. But it is not clear how you might encode the fact that the turkey is not hidden for 30 seconds, so if you shoot quickly you could still hit it. As the scenario becomes more complicated, more and more information would have to be packed into the load action rather than the hiding event. Even if this could be done, it would very unintuitive, as it would not allow you to reason about the turkey hiding except in this one case when the agent loads the gun. What if the turkey may hide for other reasons as well, or might spontaneously decide to hide, or always hides between 6PM and 7PM? Clearly, hiding is just as valid an event as any other event (such as loading), and requires similar treatment.

Given all this, let's return to the simple example as originally formulated. Taking $Deaf$ to be an atemporal predicate, the triggering of hiding by loading is captured with a conditional generation axiom:

EGEN3 $\forall e \,.\, \textsc{Load}(e) \wedge \neg Deaf \supset \exists e' \,.\, \textsc{Hide}(e') \wedge time(e) = time(e')$

To capture the simplicity of the original problem and other solutions in the literature, we assume that the hiding is simultaneous with the loading. The more natural formulation would complicate the proof as we would need to reason about whether the agent could shoot while the turkey was trying to hide.

Next, we need to add axioms for hiding events to the YTS axioms from Figure 2.7. These are:

EDEF3 $\forall e \,.\, \textsc{Hide}(e) \supset \mathit{Hidden}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e)$

EXCP3 $\forall t, t' \,.\, \neg \mathit{Hidden}(t) \wedge \mathit{Hidden}(t') \wedge t : t' \supset \exists e \,.\, \textsc{Hide}(e) \wedge \mathit{time}(e) : t'$

EXCE3 $\forall e \,.\, \textsc{Hide}(e) \supset \neg \mathit{Deaf} \wedge \exists e' \,.\, \textsc{Load}(e') \wedge \mathit{time}(e') = \mathit{time}(e)$

The EDEF3 axiom simply states that a Hide event results in the turkey being hidden. EXCP3 is a closure axiom that says that things become hidden only as a result of a Hide event occurring. Closure axiom EXCE3 is justified by the important information implicit in the problem statement that the turkey does not hide unless it hears the gun being loaded.

We then need to modify the definition of shooting (EDEF2) so that it only kills the turkey if it's not hidden:

EDEF2a $\forall e \,.\, \textsc{Shoot}(e) \supset$
$\mathit{Loaded}(\mathit{pre1}(e)) \wedge \mathit{SameEnd}(\mathit{pre1}(e), \mathit{time}(e)) \wedge$
$\neg \mathit{Loaded}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e)$

EDEF2b $\forall e \,.\, \textsc{Shoot}(e) \wedge \neg \mathit{Hidden}(\mathit{time}(e)) \supset$
$\neg \mathit{Alive}(\mathit{eff2}(e)) \wedge \mathit{time}(e) : \mathit{eff2}(e)$

Note that this characterization of a Shoot event means "the bullet leaving the gun," presumably in the direction of the turkey. It does not mean "kill the turkey," which is rather a conditional effect of the shooting (EDEF2b). The action *shoot* is "pulling the trigger" on this account.

The axioms describing the problem are then as follows (the YTS axioms plus new axioms AXH1 and AXH2):

AX0 $t0 \prec t1 \prec t2 \prec t3$

AX1 $\mathit{Alive}(t0)$

AX2 $\neg \mathit{Loaded}(t0)$

AXH1 $\neg \mathit{Hidden}(t0)$

AXH2 $\forall t, t' \,.\, \mathit{Hidden}(t) \wedge t \prec: t' \supset \mathit{Hidden}(t')$

AX3 $\mathit{Try}(\mathit{load}, t1)$

AX4 $\mathit{Try}(\mathit{shoot}, t3)$

AX5 $\forall a, t \,.\, \mathit{Try}(a, t) \equiv (a = \mathit{load} \wedge t = t1) \vee (a = \mathit{shoot} \wedge t = t3)$

Note that axiom AXH2 simply states that once the turkey is hidden it remains hiding forever. A more realistic axiomatization might allow the possibly of an UNHIDE event, and then provide explanation closure axioms for when it might occur. The details are irrelevant for purposes of the example.

To Prove: $Deaf \otimes \exists t \,.\, Alive(t) \wedge t3 \prec\!: t$

Proof: The following is a sketch of the proof, since many of the details are similar to the examples presented previously.

(a) Suppose $Deaf$. The issue is whether the turkey is hiding at the time of the shooting, *i.e.*, whether $Hidden(t3)$. Suppose it is. Then, from AX0, AXH1, and TRPT we get

$$\exists T, T' \,.\, Hidden(T) \wedge \neg Hidden(T') \wedge T : T' \wedge t3 \sqsubseteq T'.$$

Then EXCP3 gives:

$$\exists e \,.\, \textsc{Hide}(e) \wedge time(e) : T'.$$

But then EXCE3 gives $\neg Deaf$, a contradiction, since the turkey only hides if it is not deaf. Thus $\neg Hidden(t3)$.

As in the YTS, the loading succeeds (ETRY1, EDEF1) and *Loaded* persists from $\mathit{eff1}(e_1)$ through $\mathit{pre1}(e_3)$ (EXCP2a, AX5). Thus the shooting succeeds (ETRY2) and from EDEF2b and the fact that the turkey is not hiding at $t3$ (above), the turkey is dead at $\mathit{eff2}(e_3)$ where $t3 : \mathit{eff2}(e_3)$, which, assuming that reincarnation is ruled out axiomatically, rules out its being alive at any time after $t3$, as required.

(b) Suppose $\neg Deaf$. Then AX3 and ETRY1 give

$$\exists e \,.\, \textsc{Load}(e) \wedge time(e) = t1$$

and then EGEN3 gives

$$\exists e' \,.\, \textsc{Hide}(e') \wedge time(e') = t1.$$

That is, the loading succeeds and the turkey hides, since it is not deaf. This is consistent with EXCE3, and then EDEF3 yields $Hidden(\mathit{eff1}(e')) \wedge t1 : \mathit{eff1}(e')$. This persists indefinitely (AXH2), in particular until $\mathit{pre2}(e_3)$, so the shooting fails to kill the turkey (EDEF2b doesn't apply). Thus *Alive* persists indefinitely starting at $t0$ (EXCP2b, AX5), from which we can derive the desired result. □

Triggered events are very useful for characterizing many domains. For example, whenever you press a button to start a machine, you trigger an event. Our formalism allows you to define the behavior of the machine using the full power of the language to describe events. Thus, pressing one button could trigger a complex sequence of events that greatly affects the domain for extended periods of time. A simple example could involve making a simple meal of reheated pizza. The plan is to put the pizza in the microwave oven and then press the start button. While it is cooking, you take a beer out of the refrigerator and get out the dishes. When the microwave beeps, you take out the pizza and eat. The fact that the microwave is running could constrain other actions you can perform during that time. For instance, you couldn't also reheat some coffee using the microwave, as it is in use. Or if the electric service is limited, you might not be able to run the toaster oven at the same time without blowing a fuse. As simple as this scenario sounds, representing it is beyond the capabilities of most formalisms.

2.4.2 Definite Events

Definite events can be viewed as a generalization of triggered events, where there are arbitrary constraints on when the events occur. These cases are easily handled in our temporal logic. For instance, if 5AM6 is the time interval between 5AM and 6AM, then the fact that the sun will rise sometime during this time is simply:

$$\exists t, e \,.\, \textsc{SunRise}(e, t) \wedge t \sqsubset 5AM6,$$

where this event is defined by the axiom:

$$\begin{aligned}&\forall t, e \,.\, \textsc{SunRise}(e, t) \supset \\ &\quad \neg \mathit{SunShining}(\mathit{pre1}(t)) \wedge \mathit{SunShining}(\mathit{eff1}(t)) \wedge \mathit{pre1}(t) : t : \mathit{eff1}(t).\end{aligned}$$

The problem arises as to how the system can infer that the sun doesn't rise (or more importantly set) at other times than the constraints above allow. Given the information as stated, nothing would prevent this. But the problem is not with the formalism. Rather, the problem is that the above axiom doesn't capture all of what we know about the sun rising. In fact, we also know that it rises only once a day. Thus, a revised axiom is needed, and would look as follows, where $5AM6$ is as defined above, and $TODAY$ is the time interval for the entire day, which includes $5AM6$.

$$\begin{aligned}&\exists t, e \,.\, \textsc{SunRise}(e, t) \wedge t \sqsubseteq 5AM6 \wedge \\ &\quad \forall t', e' \,.\, \textsc{SunRise}(e', t') \wedge t' \sqsubset TODAY \supset e' = e.\end{aligned}$$

In other words, there is a sun rising event sometime between 5 and 6 AM, and this is the only sun rising event today.

Note that by better capturing our actual knowledge of the event, we have constructed a "closure" axiom that gives us exactly the right behavior. It would be a mistake to have tried to handle this case by some uniform strategy over all events (either by model minimization or by automatically generated closure axioms), as the appropriate axiom depends on our specific knowledge about the event. Another external event, say SchoolBusArrives, would have a different characterization as it occurs twice a day, while yet others might have more complex structure.

2.4.3 Spontaneous Events

The final class of external events to be considered involves events that might occur, in contrast to those the definitely occur as discussed above. Since the agent does not have knowledge of whether events in this class occur or not, it is possible that these events might interfere with the agent's plan any time. As a result, any minimization strategy applied to such events would result in overly optimistic plans based on reasoning similar to proof by failure: if I cannot prove that the event occurs, then it doesn't occur. Of course, this strategy eliminates the possibility of representing spontaneous events.

If an event is always possible, *i.e.*, it might spontaneously occur at any time, then this is represented by the absence of any closure axiom whatsoever. If the event has

significant effects, then it will effectively cripple the prediction process. But we would claim that this is exactly right. Consider a turkey-shoot domain again, where another agent may unload the gun at any time. In this situation, it would be impossible to construct a plan that is guaranteed to kill the turkey. This is because at every point in the plan, the other agent might unload the gun before it is fired. Without introducing some notion of probability, this seems the best that a purely logical theory can do. But interesting cases arise when events may occur spontaneously only under certain conditions: say that the other agent only can only unload the gun if the agent is nearby, or that the agent can only interfere in the morning, or that the agent can only unload the gun at most two times. Once there are some constraints on when the spontaneous events might occur, the agent once again may be able to make concrete predictions about the effects of a set of actions.

We need a minor extension to the formalism to allow us to specify constraints on spontaneous events. Specifically, a new predicate *Spontaneous* is introduced that asserts that an event instance spontaneously occured. Events that are never planned, say SNEEZE, are always spontaneous, and this is captured by an axioms of the form

$$\forall e \,.\, \textsc{Sneeze}(e) \supset Spontaneous(e).$$

Events that cannot occur spontaneously are similarly declared to not be spontaneous, as in the following axiom that says that guns never load spontaneously,

$$\forall e \,.\, \textsc{Load}(e) \supset \neg Spontaneous(e).$$

Events that may occur spontaneously, or may be caused by the agents actions will not have an axiom either way, leaving open both possibilities. More interestingly, events that may be spontaneous under certain conditions can be represented by other axioms. For instance, if the UNLOAD event may occur spontaneously only when the other agent is nearby, the appropriate axiom would be something like:

$$\forall t, e \,.\, \textsc{UnLoad}(e, t) \wedge \neg OtherNearBy(t) \supset \neg Spontaneous(e),$$

i.e., if the other agent is not nearby, then the unload event cannot occur spontaneously. This leaves the possibility open for a spontaneous unloading event under other circumstances. Note that we are encoding a notion of possibility here as a form of ignorance. An event e is spontaneous if you can prove $Spontaneous(e)$, it is not spontaneous if you can prove $\neg Spontaneous(e)$, and it is possibly spontaneous if you can neither prove $Spontaneous(e)$ or $\neg Spontaneous(e)$.

With the possibility of spontaneous events, the closure axioms developed earlier will need to be modified for events that could sometimes occur spontaneously. This is simply accomplished by adding a new disjunct to each one allowing for spontaneous occurrence. For instance, the new closure axiom for UNLOAD would be:

$$\forall t, e \,.\, \textsc{UnLoad}(e, t) \supset Try(unload, t) \vee Spontaneous(e),$$

i.e., an UNLOAD event occurs only if the agent attempts an unload action, or if it is spontaneous. If the other agent isn't nearby, then the previous axiom will eliminate the

possibility of an unload event spontaneously occuring, and it would occur only if the agent explicitly unloads it.

As an example, consider the hiding turkey scenario again. Suppose we know that the turkey may always hide spontaneously (*i.e.*, we know no restrictions on when hiding may spontaneously occur). The extended closure axiom would then be:

EXCE3 $\forall e \,.\, \textsc{Hide}(e) \supset$
$[\neg Deaf \wedge \exists e' \,.\, \textsc{Load}(e') \wedge time(e') = time(e)] \vee$
$Spontaneous(e)$

Now, the proof sketched above will not go through. Even if the turkey is deaf, we will not be able to prove that the turkey didn't go into hiding spontaneously. We would still be able to show that if it is not deaf, it will be alive after the shooting, but if it is deaf, we won't know whether it's alive or not.

Of course, the situation could be made more interesting. For instance, maybe we know that the turkey only might spontaneously hide between times $t4$ and $t7$ (say they only hide in the evening). This is easily expressed. The axiom constraining spontaneous hiding events becomes:

$$\forall e \,.\, \textsc{Hide}(e) \wedge (time(e) \prec: t4 \vee t7 \prec: time(e)) \supset \neg Spontaneous(e).$$

This axiom states that hiding is not spontaneous if it occurs before $t4$ or after $t7$ (and hence allows the possibility of its being spontaneous between $t4$ and $t7$). As a result, the original proof will go through again except for times between $t4$ and $t7$. In particular, we can again prove that the turkey will be killed if it is deaf and we shoot at time $t3$, given appropriate axioms ordering $t0$–$t7$.

It is interesting to observe that if we are only concerned with prediction, we do not need any axioms that explicitly assert when an event is definitely spontaneous, because the prediction process is driven by the closure axioms that require us to prove $\neg Spontaneous(e)$ in order to prove persistence. When considering additional reasoning tasks in the same domain, however, such as explanation, then explicit axioms stating when an event is spontaneous are very useful. This is because of the abductive nature of explanation, where explanations involving provably spontaneous axioms would require fewer assumptions than explanations that simply assume an event was spontaneous. While we have not considered explanation here, the techniques developed by Kautz (1991) for plan recognition could be adapted easily for use with our representation.

2.4.4 Discussion and Related Work

Most formalisms prohibit external events. For example, action definitions in STRIPS-based systems must encode all changes in the world, so external events are fundamentally banned from analysis. Similarly, the constructive situation calculus has the same problem—since there is no way to express information about what the state of the world is while the action is happening, there is no mechanism for allowing an event to occur while an action is being performed. Lifschitz and Rabinov (Lifschitz and Rabinov, 1989)

present a limited mechanism to handle this by allowing "miracles" to occur while an action is performed to explain why its effects are not as expected. But this does not allow external events to occur independently of actions, nor does it give events the first-class status required to represent complex situations. In addition, by minimizing the number of miracles, the approach makes it difficult to handle uncertainty about whether some external event will occur or not. Another possibility would be to encode external events as pseudo-actions so that they can be used in the result function, although they are clearly different from normal actions (*e.g.*, the agent couldn't plan to execute them).

In addition, the situation calculus typically only represents possible futures from the initial state, and has no notion of an actual future. Thus, one cannot simply state that an event E will occur, say at noon tomorrow. The obvious way to capture this is by an axiom that guarantees that every possible action sequence that extends past noon tomorrow includes E at the appropriate place in the sequence. To represent such information, you need to be able to quantify over all possible action sequences, including those where actions overlap. While the formalism can be extended in this way, one loses the constructivity that made the constructive situation calculus an attractive formalism for reasoning about action.

Even more problematic are external events that are conditional on the agent's behavior, or simply may or may not occur independent of the agent. If some event E occurs, then the reasoner must only consider situations that are constructed by an action sequence with E in it and if it doesn't occur, then E should not be in the sequence. But this forces a change in the interpretation of situations. Without external events, every situation (defined by an action sequence) indicated a possible attainable future, and to find a plan for a goal G, one needed only to find a sequence that resulting in a situation where G was true. With external events, there may be sequences that produce a situation in which G is true, but finding one does not guarantee that the goal can be achieved, since the situation may not be attainable. For example, a situation in which G is true might be constructed using an external event E, but the agent cannot guarantee that E will occur, so this situation may not be attainable.

Work based on the situation calculus that addresses external events typically rejects the constraints of the constructive approach and uses the more general formalism. In these approaches situations are arbitrary states of the world, not necessarily related to a particular action sequence. In addition, situations can be associated with times from a time line and one can quantify over situations. With this, one can introduce an $Occurs$ predicate that asserts that a particular action occurs at a specific time and that is defined by an axiom that quantifies over all situations at that time (*e.g.*, (Pinto, 1994; Miller and Shanahan, 1994)). This development moves the formalism closer to what we are proposing.

Ultimately, the main difference between our approach and these extended situation calculus representations with explicit time will probably be one of approach. We start from a representation of the actual world (or an agent's beliefs about the world), and must introduce mechanisms to allow reasoning about the effects of possible actions. The situation calculus starts from a representation based on possible futures derived from the actions, and must introduce mechanisms for dealing with information about

the actual world. While reasoning about external events is going to be complicated in any formalism, we believe that our approach is superior for several reasons. First, it allows problems to be formalized in a simpler and, we feel, more direct manner. Second, our representation is closer to the types of representations used in implemented knowledge representation systems, and thus appears more suitable for deriving practical applications, such as planners and natural language understanding systems. The main weakness is the lack of a notion of possible futures. But a modal operator for possibility is needed independently of temporal reasoning and reasoning about action. Combining such a modal operator with our linear time model would allow us to express anything that is expressible in the situation calculus. There does not seem to be any advantage to combining the two needs in a single mechanism. Interestingly, this work demonstrates that a wide range of problems can be solved without ever using the explicit modality.

2.5 Simultaneous Action

There are several levels of difficulty in dealing with simultaneous actions. The model described so far already handles some cases of simultaneous actions, namely

- When two actions cannot occur simultaneously;
- When they occur simultaneously and are independent of each other; and
- When they together have additional effects that neither one would have individually.

The more difficult case is when two actions partially or conditionally interfere. After first describing how the basic approach handles the three cases above, we will discuss some approaches to reasoning about interference.

We prefer to use a more complex domain than that of the YTS to illustrate these cases. The TRAINS domain is a transportation and manufacturing domain developed for use in the TRAINS project (Allen et al., 1995). The goal of the project is to build an intelligent planning assistant that is conversationally proficient in natural language. The domain involves several cities connected by rail links, warehouses and factories at various cities, and engines, boxcars, *etc.*, to transport goods between them. A more detailed description of the TRAINS domain and system will be presented in subsequent chapters, however a simple TRAINS domain map used in the examples to follow is shown in Figure 2.10. The important thing is that in this domain, interacting simultaneous actions and external events are unavoidable features.

2.5.1 Independent and Mutually-Exclusive Actions

When actions are independent of each other, the existing formalism does exactly the right thing. The effect of the two actions is simply the sum of the effects of actions

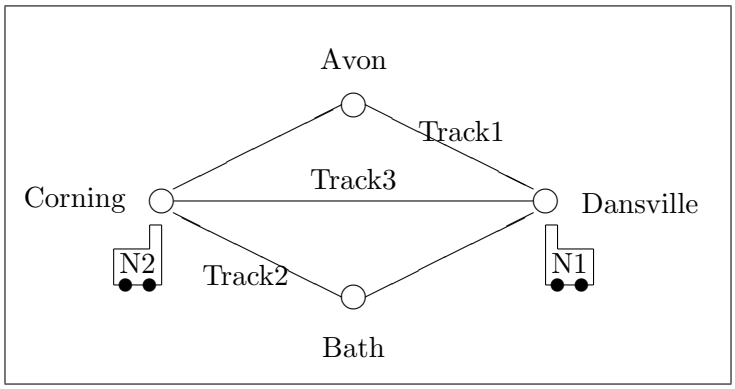


Figure 2.10: Example TRAINS domain map

individually, a result you get directly from the axioms. When there are simple interactions, such as mutual exclusion of actions under certain circumstances, this behavior typically naturally falls out of a reasonable axiomatization of the domain.

For example, consider a simplified axiomatization concerning engines moving between two cities, where n ranges over engines, $c1$ and $c2$ over cities, and x over tracks, and $move(n, c1, c2, x)$ is an action that moves engine n from $c1$ to $c2$ along track x:

ETRY1 $\forall n, c1, c2, x, t \,.\, Try(move(n, c1, c2, x), t) \supset On(n, x, t)$

ETRY2 $\forall n, c1, c2, x, t \,.\, Try(move(n, c1, c2, x), t) \wedge TrackClearFor(n, x, t) \supset$
$\exists e \,.\, \textsc{MoveBetween}(e, t, n, c1, c2, x)$

EDEF2 $\forall e, t, n, c1, c2, x \,.\, \textsc{MoveBetween}(e, t, n, c1, c2, x) \supset$
$At(n, c2, \mathit{eff1}(e)) \wedge t : \mathit{eff1}(e)$

ETRY1 states that while trying to move along the track x from city $c1$ to city $c2$, the engine n is on the track. ETRY2 states that if the track is clear when the agent attempts to move the engine, then a MoveBetween event occurs. Finally, EDEF2 states that if a MoveBetween occurs, then the engine ends up at the destination city.

In addition to these axioms, we have general knowledge about the domain that includes the fact that an object cannot be in two different places at once and what it means for the track to be clear for an engine:

AX1 $\forall n, x, y, t, t' \,.\, On(n, x, t) \wedge On(n, y, t') \wedge x \neq y \supset t \bowtie t'.$

AX2 $\forall n, x, t \,.\, TrackClearFor(n, x, t) \equiv$
$\forall n', t' \,.\, (t' \sqsubseteq t \wedge Engine(n')) \supset \neg On(n', x, t') \vee n' = n$

Note that we have not added any special knowledge about how the *move* actions interact, but these axioms already capture a good range of our intuitions about the domain. For example, consider the scenario shown in Figure 2.10 with two engines, $N1$ at Dansville and $N2$ at Corning. If $N1$ tries to move to Avon on Track 1, and $N2$ tries to move to Bath on Track 2 simultaneously, then they both get to their destinations. The scenario is axiomatized as follows:

AX3 $t0 : t1$

AX4 $Engine(N1) \wedge Engine(N2) \wedge \forall n \,.\, Engine(n) \supset (n = N1) \vee (n = N2)$

AX5 $Track1 \neq Track2$

AX6 $At(N1, Dansville, t0)$

AX7 $At(N2, Corning, t0)$

AX8 $Try(move(N1, Dansville, Avon, Track1), t1)$

AX9 $Try(move(N2, Corning, Bath, Track2), t1)$

We can prove that there is a time t, where $t1 : t$, and where both $At(N1, Avon, t)$ and $At(N2, Bath, t)$ are true.

Proof: Consider proving $At(N1, Avon, t)$. From ETRY1 and AX9 we get

$$On(N2, Track2, t1),$$

then from AX1 we get $\neg On(N2, Track1, t1)$. AX2 and AX4 then give

$$TrackClearFor(N1, Track1, t1),$$

which together with AX8 and ETRY2 gives

$$\exists e_1 \,.\, \textsc{MoveBetween}(e_1, t1, N1, Dansville, Avon, Track1).$$

Then EDEF2 gives $At(N1, Avon, \mathit{eff1}(e_1))$ and $t1 : \mathit{eff1}(e_1)$, as required. An exactly parallel proof shows that $At(N2, Bath, \mathit{eff1}(e_2))$ and $t1 : \mathit{eff1}(e_2)$, so we have shown that both actions had their intended effects. □

When these actions interfere with each other, the domain knowledge will block the unwanted inference that the events occured successfully together. For instance, consider a variant of the above scenario where both engines try to use Track 3, *i.e.*, we have:

AX8 $Try(move(N1, Dansville, Corning, Track3), t1)$

AX9 $Try(move(N2, Corning, Dansville, Track3), t1)$

In this case, we will not be able to prove that either MoveBetween event occurs. The reason is that the two move actions create a situation in which both engines are on the same track. This then prevents ETRY2 from being used, and so we can conclude very little about what events were caused by these actions.

Notice that the fact that the actions were attempted is still true. In general, we do not limit action attempts in any way, reflecting a belief that an agent may attempt most actions at any time. If the conditions are not appropriate, the actions simply won't cause the desired events. Of course, we could add additional axioms describing what happens when action attempts interact. For instance, we could add a dramatic axiom that asserts that if two move actions are attempted simultaneously on the same track, the engines crash. More realistically, we might predict a deadlock situation, or events in which the engines end up returning to their originating cities. The point here is that the temporal logic provides a fairly natural way to axiomatize the knowledge in this domain so as to capture the desired behavior.

Another form of interaction between simultaneous action attempts are resource conflicts. For instance, in the TRAINS domain, an engine requires fuel to make trips. If two engines are planning trips but there is only enough fuel at the station for one, then only one engine can succeed. Many of these cases require reasoning about quantities, which would introduce many complications beyond those necessary to make our point here. But some resources can be modeled as unitary—either you have the resource or you don't. In these cases, the techniques used above can be adapted to resources. In fact, you can view the track in the previous example as a resource required by an engine to make a trip, and the problem where the engines try to use the same track as a resource conflict.

Finally, other constraints on simultaneous action arise because of limitations on the part of the agent. While often such restrictions can be cast in terms of resource conflicts, the temporal logic also allows specific axioms about action co-occurrence. For instance, say a engineer cannot try to couple one car and decouple another at the same time. This constraint can be simply captured by the axiom:

$$\forall n, c, c', t, t' \,.\, Try(couple(n, c), t) \wedge Try(decouple(n, c'), t') \supset t \bowtie t'$$

This axiom would make a set of assertions where an agent simultaneously attempted to couple one car and decouple another inconsistent. Axioms like this can be very useful in defining simple planning systems similar to non-linear planners (*e.g.*, (Allen and Koomen, 1983)).

2.5.2 Synergistic Effects

It is also easy to define additional synergistic effects that occur when two actions occur simultaneously. For instance, you can define events that occur only when several actions are performed simultaneously. Thus, if one agent lifts one end of a piano while another agent lifts the other end, then the entire piano is lifted off the floor. Such situations are easily described in our representation. For example, assuming axioms for a *lift* action that causes LIFT events, we could specify how the simultaneous lifting of both ends of the piano results in the piano being lifted using a generation axiom such as:

$$\forall e1, e2, p, t \,.\, \textsc{Lift}(e1, t, left(p)) \wedge \textsc{Lift}(e2, t, right(p)) \supset \exists e3 \,.\, \textsc{Lift}(e3, t, p).$$

The fact that actions and events have durations allows us to make this somewhat more realistic by only requiring that the lifting of the ends overlap, rather than be simultaneous, as in

$$\forall e1, e2, p, t1, t2 \,.\, \textsc{Lift}(e1, t1, left(p)) \wedge \textsc{Lift}(e2, t2, right(p)) \wedge \neg(t1 \bowtie t2) \supset \exists e3 \,.\, \textsc{Lift}(e3, t1 \cap t2, p).$$

The function "$t1 \cap t2$" refers to the interval "common" to both $t1$ and $t2$, which must be non-disjoint (as they are here). An even more realistic version of the axiom would require that the ends were both lifted for long enough.

A more realistic example of synergistic effects involves performing an action while other events are occurring (possibly also caused by actions), to cause additional effects. Consider an example from the TRAINS world that illustrates this. To decouple a car, an engineer must activate the decoupler while the engine is moving forward. The actions of the engineer are not simultaneous here, as the engine moving forward is an event triggered by the engineer's action of setting the throttle. This scenario can be axiomatized as follows:

ETRY1 $\forall n, t \,.\, Try(setThrottle(n), t) \supset \exists e, t' \,.\, \textsc{Move}(e, t', n) \wedge t : t'$

ETRY2 $\forall n, t \,.\, Try(activate(n), t, e) \supset \exists e \,.\, \textsc{Activate}(e, t, n)$

ETRY1 states that setting the throttle triggers the moving event. ETRY2 states that the decoupler is activated only while the engineer holds down the activator button.

These two events need to occur simultaneously to uncouple a car—neither one alone results in the car being uncoupled. This knowledge is captured by a conditional generation rule that involves two events on the left-hand side. Here's a possible axiomatization:

EGEN3 $\forall e1, e2, n, c, x, t1, t2, t \,.\, \textsc{Move}(e1, t1, n) \wedge \textsc{Activate}(e2, t2, n) \wedge \neg(t1 \bowtie t2) \wedge \mathit{Coupled}(n, c, t) \wedge t : (t1 \cap t2) \supset \exists e \,.\, \textsc{Uncouple}(e, (t1 \cap t2), n, c)$

EDEF3 $\forall e, t, n, c \,.\, \textsc{Uncouple}(e, t, n, c) \supset \neg \mathit{Coupled}(n, c, \mathit{eff1}(e)) \wedge t : \mathit{eff1}(e)$

What EGEN3 says is that if the Move and Activate events overlap temporally, and if the engine and car are coupled prior to the simultaneous moving and activating, then an Uncouple event will occur (over the time during which the events overlap). EDEF3 simply states that an effect of uncoupling is that the engine and car are not coupled.

To complete this example, we would need to encode some commonsense closure axioms, such as the fact that the Move event continues until the engineer unsets the throttle. With these, we could prove that if the engineer sets the throttle and then activates the decoupler, and does nothing else, then the simultaneous occurrence of the Move and Activate events will result in the car being decoupled.

2.5.3 Interference

Complications in handling simultaneous actions and events arise when they partially interfere with each other, or interfere under certain conditions. Note that with our view of events as classifications of patterns of change, it makes no sense to talk about events interfering with each other. The same world cannot be characterized by two events whose definitions are mutually inconsistent. Interference makes sense only when talking about actions. For instance, if an agent is trying to open a door, then it is performing a *push* action hoping to cause an OpenDoor event. If, simultaneously, another agent pushes on the door the other way, then the first agent will still have performed the *push* action, but the OpenDoor event will not have occurred. Thus, we say that the two *push* actions interfered with each other. In this case, they cancelled each other out. Of course, action attempts might also be affected by external events that are occurring simultaneously. We will consider both cases below.

When an action interacts with other events that are occurring, the interaction can be treated much the same as we handled conditional effects of action attempts when different properties hold. This technique formalizes the problem in terms of events, and the different effects result from synergistic interaction between events. For example, consider a situation where we want to load a boxcar from an automatic hopper. If we push the start button, the hopper tips and dumps its cargo into the boxcar. This, of course, requires that the boxcar be in the appropriate location under the hopper. If the boxcar was not under the hopper to start with, it is clear that the loading of the car would not occur although the hopper would still be emptied. The more interesting case

occurs when the boxcar is initially under the hopper, but then moves away while being loaded. Intuitively, we say that the moving of the boxcar interferes with the loading of the car. Again, the hopper would still end up empty, but the boxcar would not contain all the cargo. Here's a simple axiomatization of these actions and events.

First, pushing the start button causes an EMPTYHOPPER event to occur, and occurrence of an EMPTYHOPPER event entails that the hopper is empty afterwards:

ETRY1 $\forall h, t \,.\, Try(startHopper(h), t) \supset \exists e, t' \,.\, \textsc{EmptyHopper}(e, t', h) \wedge t : t'$

EDEF1 $\forall e, t, h \,.\, \textsc{EmptyHopper}(e, t, h) \supset Empty(h, \mathit{eff1}(e)) \wedge t : \mathit{eff1}(e)$

In addition, we need another axiom about moving that captures fact that when a MOVE event is occurring, the car is continually changing position:

AX1 $\forall e, t, c, t1, t2 \,.\, \textsc{Move}(e, t, c) \wedge (t1 \sqsubset t) \wedge (t2 \sqsubset t) \wedge (t1 \prec: t2) \supset$
$\exists l1, l2 \,.\, l1 \neq l2 \wedge At(c, l1, t1) \wedge At(c, l2, t2)$

And, of course, you cannot be two place at the same time:

AX2 $\forall o, l, l', t, t' \,.\, At(o, l, t) \wedge At(o, l, t') \wedge l \neq l' \supset t \bowtie t'.$

Finally, the information that the boxcar must be under the hopper to be loaded is cast as an event generation axiom, followed by an axiom defining loading events:

EGEN2 $\forall e, t, c, h \,.\, \textsc{EmptyHopper}(e, t, h) \wedge At(c, h, t) \supset \exists e' \,.\, \textsc{Load}(e', t, c, h)$

EDEF2 $\forall e, t, c, h \,.\, \textsc{Load}(e, t, c, h) \supset \mathit{Loaded}(c, \mathit{eff1}(e)) \wedge t : \mathit{eff1}(e)$

We think this is an uncontroversial characterization of some of the general knowledge an agent would need to reason about this domain. And this is all that is needed to reason about the interaction. For instance, consider a situation in which the agent pushes the start button when the boxcar is under the hopper, but the boxcar is moving. Under suitable assumptions about the duration of events, the definition of the MOVE event can be used to prove that there is a time during the EMPTYHOPPER event when the boxcar is not under the hopper. Thus, a LOAD event cannot be inferred from axiom EGEN2. Of course, this is a crude axiomatization for the purposes of illustrating the technique. A better characterization would have axioms that enable reasoning about the changes that occur during the loading action. While this can be done, it doesn't add to the point being made here.

It may be true that we could characterize all interactions between simultaneous actions in terms of events in this way. Ultimately, however, this would require reducing everything to events characterizing physical force equations. In fact, this is what is suggested in many papers addressing simultaneous action (*e.g.*, (Pednault, 1986)). We do not believe this is appropriate for a commonsense theory of action, as it requires agents to understand the physics of the world at a fairly deep level in order to understand simple everyday interactions.

Good examples that show a need for another method can be found in characterizing an agent's knowledge about how its own action attempts interact when performed simultaneously. For instance, a two-armed robot might know that (1) it can lift either a large or a small block individually, (2) it can lift two small blocks simultaneously, but (3) if it tries to lift two large blocks, it will tip over. This knowledge is captured directly in the following axioms:

ETRY1 $\forall b, t \,.\, Block(b) \wedge Try(lift(b), t) \wedge$
$\neg\exists b', t' \,.\, Overlaps(t', t) \wedge Try(lift(b'), t') \supset \exists e \,.\, \textsc{Lift}(e, t, b)$

ETRY2 $\forall b1, b2, t \,.\, SmallBlock(b1) \wedge SmallBlock(b2) \wedge$
$Try(lift(b1), t) \wedge Try(lift(b2), t) \supset$
$\exists e1, e2 \,.\, \textsc{Lift}(e1, t, b1) \wedge \textsc{Lift}(e2, t, b2)$

ETRY3 $\forall b1, b2, t \,.\, LargeBlock(b1) \wedge LargeBlock(b2) \wedge$
$Try(lift(b1), t) \wedge Try(lift(b2), t) \supset \exists e \,.\, \textsc{TipOver}(e, t)$

AX1 $\forall e, e', t, t' \,.\, \textsc{Lift}(emt) \wedge \textsc{TipOver}(e', t') \supset t \bowtie t'$

Now, some will criticize this solution, saying we just wrote out the answer. But this is exactly the knowledge that an agent would acquire after a short amount of experimentation in the domain. Also, it is reasonable to assume that the agent has relatively complete knowledge of what actions it will attempt over a certain period of time, so negative conditions such as the one used in ETRY1 would be easy to prove. In fact, Haas (1992) has used this idea to build a reactive planner that uses explanation closure. In any event, a formalization at this level seems more more plausible than a solution that solves this problem by reasoning about force equations, which would require much more detailed knowledge of the domain. As well, such a solution might well give answers at the wrong level of detail for them to be useful in a commonsense reasoner.

We could also use something like McCarthy's *Ab* predicate (McCarthy, 1980) in ETRY1, which would allow us to incrementally add conditions that entail an abnormal lifting attempt. However, if we used a standard nonmonotonic method to minimize *Ab*, we would risk losing information about cases where the robot is uncertain. For instance, the axioms above say nothing about what would happen if the agent simultaneously lifts a small block and a large block. This reflects the fact that the agent doesn't know. With a technique based on predicate minimization (such as (Lin and Shoham, 1992)), the agent would assume that it would not tip over, possibly a dangerous assumption. We would instead use explanation closure techniques on the *Ab* predicate which allow us to leave such cases uncertain.

The agent might also have knowledge about how its actions interact with external events. For example, consider an agent's knowledge about driving a car. The action of turning on the ignition starts the event of the engine running:

$$\forall c, t \,.\, Try(turnIgnition(c), t) \wedge level(c, t) > 0 \supset$$
$$\exists e, t' \,.\, \textsc{RunEngine}(e, t', c) \wedge t : t',$$

where the function $level(c, t)$ produces the minimum level of the gas during time t. When the engine is running, pressing the accelerator causes the car to move forward:

$$\forall c, t \,.\, Try(pushAccelerator(c), t) \wedge \\ (\exists e, t' \,.\, \textsc{RunEngine}(e, t', c) \wedge t \sqsubseteq t') \supset \exists e' \,.\, \textsc{Move}(e', t, c).$$

These two axioms capture the interaction between the agent's action and a simultaneously occurring external event directly without requiring a deep understanding of the underlying causal model of how cars work. In addition, the agent might have other knowledge about the engine running event, such as the fact that it requires gasoline in the tank, and consumes gasoline as it runs:

$$\forall e, t, c \,.\, \textsc{RunEngine}(e, t, c) \supset level(c, t) > 0 \wedge \\ \forall t1, t2 \,.\, (t1 \sqsubset t) \wedge (t2 \sqsubset t) \wedge (t1 \prec: t2) \supset level(c, t1) > level(c, t2).$$

Note that this last axiom could not be expressed if the RunEngine event were formalized as a static property that is a precondition for the moving the car. Thus, to capture all this information in a single framework, the engine running should be formalized by an event.

2.5.4 Discussion and Related Work

For the most part, treatments of simultaneous actions in the literature are limited. As mentioned earlier, STRIPS-based systems (*e.g.*, (Tate, 1977; Vere, 1983; Wilkins, 1988)) only allow simultaneity when the actions are independent. A few others (*e.g.*, (Pednault, 1986)) allow for synergistic effects cast in terms of domain constraints on states (*e.g.*, in any state s, if the left side of the piano is lifted, and the right side is lifted, then the piano is lifted).

The situation calculus shows more promise with the introduction of action composition operators. For instance, given two actions a_1 and a_2, then the action $a_1 + a_2$ is the action of performing the two simultaneously (*e.g.*, (Gelfond et al., 1991; Schubert, 1990)). Explicit axioms can then be given for these complex actions, and mechanisms can be introduced to automatically derive such axioms from the individual actions if they are independent of each other (*e.g.*, (Lin and Shoham, 1992)). If the actions are not independent of each other, some reasonable solutions can be found and, as long as actions are instantaneous, it appears that the theory can remain constructive. But these approaches do not seem to be easily extended to handle the more complex cases in which actions have duration and may be temporally related in complex ways. Pelavin (1991) contains an extensive analysis of the problems that arise in general with simultaneous interacting actions.

Some work has explored the possibility of allowing a duration for each action (Gelfond et al., 1991). Then the duration of a sequence of actions would simply be the sum of the individual action durations. But there is little point to doing this unless one allows these extended actions to overlap with other actions and external events. If these cases are not allowed, adding the durations adds little additional expressive power as

the problems that can be stated with durations are effectively isomorphic to problems without the durations. But it is not clear how overlapping and interacting actions can be handled. For instance, if a temporally extended action A maps a situation s to a situation $A(s)$ after a delay of n seconds, how would this combine with another action B that starts 3 seconds after A begins, where B maps a situation s to $B(s)$? If the axioms for A are not conditional, then we end up in state $A(s)$ no matter whether action B occurs or not. If the axioms for A depend on whether B occurs or not, what is the final situation? Clearly, $A(B(s))$ is not right, as it implies that B was executed first and A second, and $B(A(s))$ makes similarly bad claims the other way. Of course, since the situation calculus is embedded in first-order predicate calculus, and we have introduced time into the representation, we should be able to express some axioms that would characterize the what happens when B occurs 3 seconds after the start of A. But by the time we introduce all the temporal mechanisms, it is not clear whether the mechanism of the original situation calculus helps matters, or just gets in the way.

Our approach provides a range of techniques for representing information about interacting actions. In cases where one has detailed knowledge of the causal structure of a domain, the problems can usually be reduced to a level of independent events, and knowledge about interactions are captured by event generation axioms. In other cases, it is more natural to have explicit knowledge about how certain actions interact with events. We believe this flexibility is important for building comprehensive knowledge bases that contain information about a wide range of situations and that are applicable for different reasoning tasks. In addition, we have shown that we can handle these complex cases without introducing any new mechanisms beyond the basic logic discussed in Section 2.2.

2.6 Future Work

One of the most significant open issues regarding our approach has to do with temporal durations. A realistic characterization of almost all the examples in this chapter would require such a capability. For instance, it is not realistic to say that if an agent tries to turn the ignition on the car for any length of time, then the engine will start. If the action is tried for too short a time, the engine probably won't catch. And if the action is tried for too long a time, the starting motor will burn out. So durations play a critical role. At first glance, adding durations does not pose a problem. One simply defines a function that, given an interval, returns a value on some metric scale. A small number of axioms are required to define the appropriate properties of this function. We have not pursued this here as it would further complicate the examples and remove attention from our main points, and in any case, this simple metric model doesn't solve the problem. Consider the action of starting the engine again. There is no minimum time or maximum time within which the engine is guaranteed to start. The duration required depends on the condition of the car, the weather, and many other factors that the agent simply won't have access to. A better formalization of the durational constraints would be that the agent turns the ignition until the engine starts, or a certain time elapses and the agent gives up for fear of burning out the motor, or

the battery runs flat. The logic we propose offers no direct solution to this problem, and it remains an important challenge.

Another important issue is the introduction of probabilistic knowledge. By staying within standard first order logic, for example, we are restricted to saying that an event will definitely occur, or that it might possibly occur. We cannot say that an event is very likely to occur, or that it is very unlikely to occur. Such knowledge is crucial for predicting the likely effects of actions and thus for evaluating the likelihood of success for proposed plans. Further, since all formalisms must make assumptions, the ultimate evaluation should be based on how likely the assumptions are to hold. This is another motivation for favoring the explanation closure technique. It makes the assumptions that are made explicit, and thus potentially available for probabilistic analysis. Some initial work on this is described in (Martin, 1993; Martin and Allen, 1993). It is much more difficult to see how techniques that build the assumptions into the semantic model could be extended to support probabilistic reasoning.

Finally, it needs to be acknowledged that formalizing knowledge using the more expressive temporal representation can be difficult. Subtle differences in meaning and interactions between axioms may be more common than in less powerful representations, and more experimentation is needed in building knowledge bases based on our representation. But it seems to us that this is the price to pay if we want to move beyond Yale Shooting and the Blocks World. Our representation is based on intuitions about the way people describe and reason about actions in language, which we believe makes it more natural, intuitive, and grounded in common sense, and hence more suitable for building intelligent interactive systems that reason about actions and events.

2.7 Summary and Contributions

Many of the techniques described in this chapter have appeared previously in the literature during the development of the interval temporal logic representation. What this work has done is to combine them into a unified and, we feel, intuitive and straightforward (given the complexity of the phenomena) framework. The logic has the following features:

- It can express complex temporal relationships because of its underlying temporal logic.
- It supports explicit reasoning about action attempts and the events they cause, which allows explicit reasoning about success and failure of attempted actions, and subsumes work on conditional effects.
- It handles external events in a natural way, making only minimal distinctions between external events and events caused by the acting agent.
- It handles simultaneous actions in a direct manner, including cases of simultaneously interacting actions.

By using an explicit temporal logic with events, we have been able to handle all these problems within a standard first-order logic. We believe this formalism to be more powerful than competing approaches. We also believe that it is simpler than these other formalisms will be once they are extended to handle the temporal complexities of realistic domains, assuming they can be successfully extended.

The specific contributions documented here are:

- To revisit the logical foundations of the interval temporal logic representation and clarify some of the formal details, such as the issue of discrete variation and the TRPT theorem used extensively to reason about change.
- A clarification and elaboration of the approach to causal reasoning based on, *e.g.*, (Allen and Koomen, 1983) or (Allen, 1991a). This includes the uniform use of role functions and the semantics of the Try predicate.
- Adaptation of the explanation closure technique to the temporal logic approach, which adds support to the argument that these explicit "frame" axioms produce an attractive and viable theory. Formally, they allow us to stay within first-orer logic rather than introducing a specialized semantic theory that we would have to validate independently. Practically, the knowledge captured in these axioms is knowledge that agents that do more than predict the effects of Yale Shooting will need anyway. The fact that the assumptions are explicit in this approach is an important fact that we will return to in the next chapter when we consider reasoning about the assumptions underlying a plan.
- Application of the logic and explanation closure to the Sandewall test suite and comparison with other approaches based on nonmontonic reasoning techniques such as circumscription.
- Reconsideration of external events and simultaneous actions within the new, unified framework and in terms of problems of interest to us in building realistic interactive reaosning systems.

With the formal background and intuitions described in this chapter behind us, we can now turn our attention to applying the representation to reasoning about plans for mixed-initiative planning.

3 Plan Representation and Reasoning for Mixed-Initiative Planning[1]

The previous chapter described an approach to reasoning about time, events, and actions within a given scenario. I now turn to the question of representing plans and reasoning about them. One of the issues we will have to address early on is what, exactly, is a plan. The answer, I will claim, is significantly more than simply a set of actions to be performed, at least if we want to do more than simply construct plans in in artificial, over-simplified domains.

Viewing plan reasoning as goal-directed reasoning about action, I am interested in the many cognitive tasks that make up this kind of reasoning: plan construction (planning), plan recognition, plan evaluation and comparison, plan repair (replanning), and the like. I feel that any representation of plans should be able to accomodate all these sorts of reasoning, and not be specialized to one of them at the expense of the others. To emphasize this aspect, I will concentrate on plan reasoning in the context of cooperative, interactive systems. In particular, this includes so called "mixed-initiative" or "intelligent assistant" systems which are intended to interact with people in a natural manner. But I claim that the same considerations apply to less ambitious systems with, say, graphical front-ends, and even to autonomous agents that need to coordinate their activities to achive their goals.

This chapter is organized as follows. We begin by considering the problem of mixed-initiative planning and describing data gathered from experiments with actual mixed-initiative planning scenarios. This leads to several conclusions about plan representation and reasoning that motivate my approach. One of these is the need for an expressive and natural underlying representation of events, actions, and time such as that presented in the previous chapter. Another is the fundamentally defeasible nature of reasoning about plans in mixed-initiative interaction. I therefore describe a formal system of defeasible reasoning based on argumentation, and then apply the formalism to reasoning about plans by seeing plans as arguments that a certain course of action under certain conditions will achieve certain goals. This includes a discussion of causality and qualifications, persistence reasoning based on explanation closure assumptions, and formal properties of plans-as-arguments. To relate this to traditional planning, I then present a reconstruction of one formal model of STRIPS planning within the defeasible reasoning

[1]Some aspects of the work presented in this chapter were presented previously at the Second International Conference on AI Planning Systems (Ferguson and Allen, 1994).

paradigm. Since this work is really represents first steps towards a full theory of plan representation, I present a fairly detailed examination of related work and issues for future consideration. I conclude with a summary of the claims and contributions of this part of the dissertation. The next chapter describes the implementation of these ideas in the TRAINS system domain plan reasoner.

3.1 Mixed-Initiative Planning

As with the representation of time, events, and actions presented in the previous chapter, my aim for a representation of plans is significantly broader than most AI formalisms. I are interested in *mixed-initiative planning*, where more than one agent can contribute the plan or plans under consideration. This is different from traditional planning or even from multi-agent planning, where a single planner is constructing plans for execution by multiple agents. In mixed-initiative planning, contributions to the plan come from multiple sources via some form of communication. That communication can be explicit, as in a natural language or graphical front-end, or implicit from an agent's observations of other agents' behaviours. I claim that this makes mixed-initiative planning significantly more complex than can be accounted for by traditional models of planning and plan recognition.

On the other hand, mixed-initiative systems are interesting because they at least partially embody the AI dream of intelligent, interactive agents in the sense envisioned by Turing. My work on mixed-initiative planning is ultimately aimed at developing such a comprehensive, interactive, intelligent planning assistant, the TRAINS system, which can communicate in a variety of modalities including natural language. Details of the current implementation of the TRAINS system are presented in Chapter 4 as part of the description of the TRAINS plan reasoner that we have developed based on the ideas presented in this and the previous chapter.

Prior to building the system, however, we (I and others in the TRAINS project) conducted a series of experiments to gather examples of mixed-initiative planning phenomena in order to guide the design and implementation of the system. We recorded approximately thirteen hours of dialogues between two people, one playing the role of the system and the other playing the manager, as they collaborated in building a plan for a goal put to the manager. The domain of discourse is a transportation world with cities, rail links between them, engines, boxcars, and the like to move things around, and a variety of commodities. The participants could not see each other and communicated solely through headsets. They each had a map of the TRAINS domain, shown in Figure 3.1. These recordings have been transcribed and annotated in various ways, including prosodic marking and part-of-speech tagging, and supports a variety of research projects based on TRAINS (*cf.* (Allen et al., 1995)). The transcripts are available in (Gross et al., 1993).

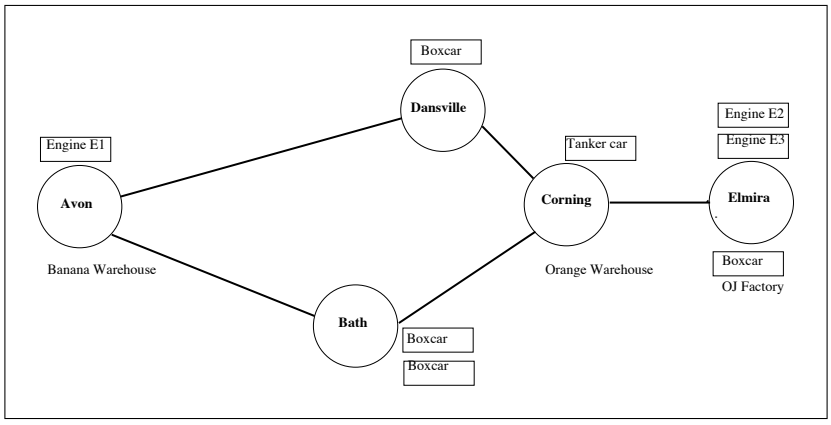


Figure 3.1: TRAINS world map for dialogue collection

3.1.1 Sample Mixed-Initiative Planning Scenario

I would like to consider one of the collected dialogues in order to highlight the roles of plan representation and reasoning in mixed-initiative planning. It should be noted at the outset that although these dialogues involve complicated natural language phenomena such as partial utterances, speech repairs, turn-taking, definite reference, *etc.*, the complexity of the interaction is not due solely to the fact that the interaction is in natural language. Rather, I believe that such interaction is a fundamental component of mixed-initiative planning in any modality, a point I will return to shortly.

The manager begins the dialogue by stating her goal:

```
M: We have to ship a boxcar of oranges to Bath by 8 AM and it is
   now midnight.
```

Immediately, this illustrates an important aspect of mixed-initiative planning, namely the fact that the participants may have different abilities, beliefs, and preferences. Typically in dialogues such as ours, the manager has goals she wants to satisfy and the system has detailed (although imperfect) knowledge about the world and about other plans that might be in progress.

The system acknowledges the manager's statement and the manager proceeds to start working on her plan:

```
S: Okay.
M: Okay. Um... all right. So, there are two boxcars at ... Bath
   and one at ... Dansville and ...  [4sec]
```

Presumably this is also an indirect suggestion to use one of these boxcars to ship the oranges. Just as the system is about to offer a suggestion, however, the manager interrupts and a clarification subdialogue ensues:

```
S: And there's ...
M: [interrupts] Wait. I've forgotten where the oranges are.
   Where are the oranges?
S: The oranges are in the warehouse at Corning.
```

In addition to clarifying the situation for the manager, there is presumably also an indirect suggestion to ship those particular oranges. Even if the manager's subsequent utterances reject this interpretation, the system needs to have understood the possible role of the oranges in the plan in order to understand the rejection.

The manager then specifies an action to be added to the plan, which the system accepts:

```
M: Okay. So we need to get a boxcar to ... Corning.
S: Right.
M: All right. So why don't we take one of the ones from Bath?
S: Okay.
M: So ...
```

Now, at this point the system has presumably recognized the manager's plan up to here, and then does some planning to elaborate it (perhaps keyed by the manager's filled pause), identifies an ambiguity that it can't resolve, and so asks the manager:

```
S: [interrupts] We need ... Okay, which engine do you want to
   bring it?
```

This illustrates another important aspect of mixed-initiative planning: the ability of all participants to ask questions. Previous AI systems have generally been expected to be relatively omniscient, at least within a tightly circumscribed domain. In mixed-initiative systems, the system can use the fact that it is participating in a conversation to ask a question rather than make an uninformed decision or spend excessive time computing an answer. This requires that the system understand and be able to reason about its limitations in these respects.

The conversation then proceeds through an evaluation of the alternatives which is eventually cut off when the manager asks what might be interpreted as a domain meta-query:

```
M: Oh. How about ... Well, Let's see. What's shorter the distance
   between Avon and Bath or Elmira? It looks like it's shorter
   from Elmira to Corning, so why don't you send E2 to Corning?
S: Okay.  [4sec]
M: In fact ... What did I say? Did I say ... Did I send a boxcar
   from Bath to Corning? It also looks like it's shorter from
   Dansville to Corning, so why don't you send that boxcar to ...
   Corning ...
S: [interrupts] Okay, with ...
M: [interrupts] instead ...
S: [interrupts] with engine E2.
M: Yes.
S: Okay, so you want ...
M: [interrupts] Oh wait. Okay, I ... All right, I misunderstood.
   So you have to have an engine in order to move a boxcar,
   right?
```

In addition to various complicated linguistic phenomena (*e.g.*, speech repairs, interleaved interruptions, reference patterns), this exchange illustrates the way participants in mixed-initiative planning often use plans as a framework within which to explore alternatives rather than as direct guides to action. That is, they form plans to weigh their options prior to committing to action and as a consequence will form many more plans than they will ever execute. In some cases it may not even matter if the plans are executable, if they are exploring alternatives along some other dimension or at a different level of abstraction. This *hypothetical* use of plans will be an important motivation for the formal development that follows.

In any case, the system answers the query, only to have the manager come back with an even more complicated meta-query about the abilities of the agents and their

relationship. She then continues on with a more concrete query which the system interprets as suggesting an action in the plan:

```
S: Right.  [3sec]
M: What's ... Is it shorter to move an engine from Elmira?
   [3sec] Can you figure things out for me? Can you figure out
   what's the shortest? [3sec] What would be faster: to send an
   engine from Elmira to ... one of the boxcars or from Avon?
S: Well there's a boxcar already at Elmira [3sec] and you mean
  to go to Corning.
```

Having recognized the suggestion to use the engine to move a boxcar, the system has to do some plan recognition to fit it into the plan and then evaluate alternatives, in this case leading to the suggestion of an alternative instead.

With that clarification, the rest of the dialogue is a straightforward summary of the plan and final confirmation (except for the tricky "What am I allowed to do?"):

```
M: Yes. All right. I didn't see that boxcar. Okay. Great. Send
   E2 with the boxcar from Elmira to Corning.
S: Okay. [2sec] Okay. We can do that.
M: All right. And [3sec] once it ... gets there [2sec] What am I
   allowed to do? Okay. I can make a whole plan at this point.
   Right?
S: Yeah.
M: Okay. Once it gets there ... have it fill up with oranges and
   go straight to Bath. [2sec]
S: Okay. Then we get to Bath at ... 5 AM.
M: Great.
S: So we're done.
```

Before discussing the lessons we have drawn from data such as this dialogue, it is worth repeating that this complicated exchange was necessary for the construction of even this simple plan for a single goal with no complications. In fact, this scenario was used as a warmup exercise for the participants, who were then given a more complicated goal and a scenario in which various things went wrong during the mixed-initiative planning process. Although the planning required to "solve" the problem may be trivial (at least in some models), it is hard to imagine a traditional planner participating in this sort of dialogue.

3.1.2 Lessons from Mixed-Initiative Planning

What can we conclude from consideration of dialogues such as the one just presented about what makes mixed-initiative planning a new and interesting area for research? To what extent are existing approaches to planning and other plan reasoning tasks applicable and what do I propose to address the new issues?

First, the most obvious lesson to be drawn is that mixed-initiative planning is fundamentally a process of *communication*. That is, there is planning and plan recognition, *etc.*, being performed, but what is interesting is that it is all being performed in a communicative context. If we ignore the communicative aspects of the process we will be unable to build systems that can participate in realistic mixed-initiative environments. As I will discuss in Section 3.5.3, this perspective is shared by many of the researchers interested in collaborative discourse but has generally not been appreciated by the planning community.

I should stress that this communicative aspect is independent of the modality in which the communication is performed. That is, while the preceding dialogue clearly illustrates the conversational aspect of mixed-initiative planning in natural language, I believe that similar phenomena would arise in other modalities. For example, an effective graphical user interface that interfaces with an underlying plan reasoner would need to express the same sorts of concepts. These include things like focus of attention, direct and indirect reference to objects, specification of parts of a plan, like goals, actions, and assumptions, and relationships between parts of a plan, such as enablement or doing something for a particular purpose. In fact, the plan reasoning agent need not be interacting with humans at all. A group of autonomous agents that communicate in some machine-oriented language and that need to coordinate or at least take account of the actions of other agents will need to reason about exactly these same things. And in fact, even what appear at first glance to be strictly natural language phenomena, such as resolving references to objects, will arise for realistic autonomous agents that have only limited perceptual and internal resources. The lesson is that it is not the modality of communication that matters, but rather its complexity.

The second lesson to draw from our study of mixed-initiative planning is that it is fundamentally based on defeasible reasoning, that is, conclusions are subject to revision given new information or more time to reason. There are several reasons for this. First, deriving from the previous lesson, the opacity and ambiguity of communication makes reasoning based on communicated information defeasible. Not everything can be communicated (opacity) and what does get communicated can usually serve several purposes in the plan reasoning process (ambiguity). And, as noted above, this is the case not just for natural languages but also for graphical interface languages or an autonomous agent language. Second, there is the real-time nature of interactive behaviour. Utterances are cut off, suggestions modified or retracted, and it's simply not going to be feasible to replan from scratch all the time, even if you could figure out what "from scratch" was in that context. On the other hand, given more time to think, we would hope that the agent's reasoning would improve, a hallmark of defeasible reasoning. Note that this form of defeasibility (nonmonotonicity in computation) is different from the defeasibility based on new evidence studied in most of the the work on nonmonotonic reasoning, a point we will discuss in Section 3.2.4. Third, there are more common sources of defeasibility, such as incomplete knowledge of the world, uncertain effects of actions, and the like. For systems that need to interact with people, they are going to have to represent the world at a level that seems natural to us, which includes these uncertainties. Similarly, autonomous mixed-initiative systems will have the uncertainties thrust upon

them by the world and their sensors. Finally as regards defeasibility, people (or other agents) can be wrong. It is important that the system be able to represent and reason about incorrect plans, both those of the user and its own that it has subsequently lost faith in. The lesson here is that the ability to change one's mind is an important aspect of collaborative reasoning, otherwise why bother collaborating?

Putting these two lessons together, the communicative nature of mixed-initiative planning and its basis in defeasible reasoning, I can start to formulate a new view of plans and their role in intelligent behaviour. What gives a plan conclusive force in this view is not a deductive guarantee of correctness as with almost all existing formal models of plans and planning. Rather, a plan is "good" to the extent that it can convince us that its execution will result in its goals being achieved. That is, a plan is an *argument* that executing its actions will result in its goals being achieved, given the assumptions on which it is based. The "us" being convinced can be the planning agent itself reasoning dialectically, or it can be another agent engaged in a dialogue with the planning agent, as in my example of mixed-initiative planning.

This characterization immediately leads to several desirable properties. For example, agents might have different criteria against which plans are evaluated, without the plans needing to be incoherent if they disagree with an agent's knowledge (as with STRIPS plans when the preconditions of an action do not hold). Also, plans need only be specified as far as necessary to convince the agent or agents of their feasibility (which may vary from agent to agent of course). Agents can incrementally refine their plans just as arguments are refined in response to criticisms or perceived deficiencies. And plans seen as arguments can be ranked and compared along many possible dimensions. The rest of this chapter takes this intuition that plans are arguments about action seriously and develops a representation of plans based on explicit reasoning about arguments.

3.2 Argument Systems

In this section I present a formal description of a defeasible reasoning system based on those of Loui (1987) and of Pollock (1987; 1992a). This is a formal system whereby defeasible conclusions are drawn from an initial set of facts and defeasible rules. The reasoning is both *nonmonotonic*, whereby conclusions may be retracted in light of new information, and *ampliative*, meaning that conclusions can change given more time to reason. I will return to this point at the end of the section when we compare defeasible reasoning based on argumentation to other approaches.

3.2.1 Basic Definitions

I start with a logical language $\mathcal{L}$ together with a derivability relation $\vdash$.

Definition 1 A DEFEASIBLE RULE is a pair $\langle \Phi, p \rangle$, written "$\Phi \rightarrow p$," where Φ is a set of propositions (the PREMISES of the rule) and p is a single proposition (the CONCLUSION of the rule).

A defeasible rule is to be read something like "Φ is reason to believe p," or "The truth of Φ lends support to the truth of p," or, in a statistical setting, "Most Φ's are p."

Definition 2 A DEFEASIBLE KNOWLEDGE BASE is a pair $\langle KB, \Delta \rangle$ where KB is a set of propositions (facts) and Δ is a set of defeasible rules.

For example, consider the following defeasible knowledge base:

$$\begin{aligned} \Delta &= \{B(x) \rightarrow F(x)\ ,\ P(x) \rightarrow \neg F(x)\} \\ KB &= \{\forall x\ .\ P(x) \supset B(x)\} \end{aligned}$$

That is, bachelors are typically frustrated, priests are typically not frustrated, but by definition all priests are bachelors. The variables in the defeasible rules are schematic.

Agents reason from their defeasible knowledge bases by constructing *arguments*, as follows.

Definition 3 An ARGUMENT STEP is a pair $\langle \Psi, q \rangle$, written "$\Psi \rightarrow q$," where Ψ is a set of propositions (the PREMISES of the step) and q is a single proposition (the CONCLUSION of the step).

Definition 4 An ARGUMENT is a set of argument steps $\{\langle \Psi_i, q_i \rangle\}$.

Definition 5 The PREMISES of an argument A are those propositions in the conclusions of steps with empty premises, *i.e.*, $Premises(A) = \{q | \langle \emptyset, q \rangle \in A\}$. The CONCLUSIONS of an argument A are the premises of steps with empty conclusions, *i.e.*, $Conclusions(A) = \bigcup\{\Psi | \langle \Psi, \epsilon \rangle \in A\}$.

I require that arguments be well-founded, that is, that there are no steps $\langle \Phi, p \rangle$ and $\langle \Psi, q \rangle$ in A such that $p \in \Psi$. In this case, arguments correspond to directed, acyclic graphs uniquely labeled with propositions, such as illustrated in Figure 3.2. An argument step $\langle \Phi, p \rangle$ corresponds to a node labeled by p with children each labeled by an element of Φ. The sources of the graph correspond to $Premises(A)$, the sinks to $Conclusions(A)$. We will refer to the set of propositions used in an argument by $Props(A)$.

Continuing the previous example, we can form the following two arguments:

$$\begin{aligned} &P(fred) \rightarrow \neg F(fred) \\ &P(fred) \rightarrow B(fred) \rightarrow F(fred) \end{aligned}$$

Both of these arguments have the same premises, namely that Fred is a priest, but they differ in their conclusion as to whether he is frustrated.

Note that although the definitions of defeasible rules and argument steps are identical, I have not yet placed on restrictions on the relationship between arguments and an agent's defeasible knowledge base. The following definition provides the connection.

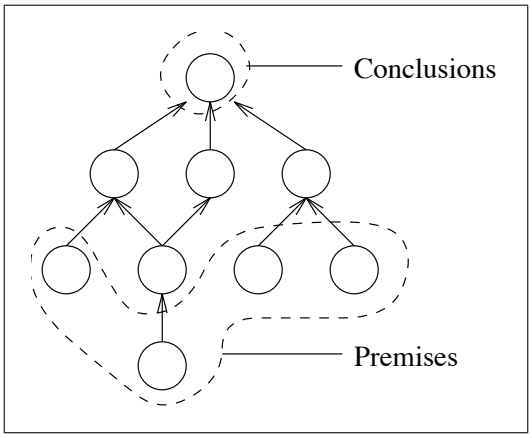


Figure 3.2: Graphical representation of an argument

Definition 6 An argument A is WELL-FORMED with respect to $\langle KB, \Delta \rangle$ if every step in A with the exception of members of $Premises(A)$ is either an instance of a rule in Δ or corresponds to the underlying derivability relation (*i.e.*, $\langle \Psi, q \rangle \in A$ if $\Psi \vdash q$).

In the previous example, both arguments about Fred being frustrated are well-formed.

Some comments regarding well-formed arguments are in order. First, it is important that we allow arguments that are not compatible with a particular defeasible knowledge base, since we might be in the position of recognizing or evaluating the arguments of another agent who might have a different defeasible knowledge base. Of course, in constructing our own arguments, we will typically only be interested in well-formed ones, but this is an added constraint and not part of the definition of an argument. Previous work on argument-based reasoning has been concerned solely with forming well-formed argument, but for my purposes we need to allow ill-formed arguments to nonetheless be arguments.

Second, the issue of deductive closure in argumentation (*i.e.*, whether to allow argument steps corresponding to the underlying derivability relation) is something of an open question. Loui (1987) includes it. Pollock (1992a) does not, preferring explicit inference rules and no underlying reasoner. Lin and Shoham (1989) (about which more later) include it, but comment that an alternative would be to include any finite axiomatization of first-order logic. My feeling is that it is intuitive to have underlying reasoners of various kinds whose conclusions can be used in arguments without recording all the details of the underlying derivation. In this sense, arguments express the "important" parts of the reasoning. The definition above does not preclude well-formed arguments that do list explicitly every detail of the reasoning, it just allows "shortcuts" relative to the underlying reasoner.

Note that I exempted the premises of an argument from the definition of well-formedness. This is because they play a special role in the evaluation of arguments to be described later.

3.2.2 Conflict and Defeat

It should be clear from this that arguments are similar to derivations or proofs in deductive logic, with defeasible rules taking the part of inference rules. But whereas the existence of a derivation ensures the truth of the conclusion (for a sound proof procedure, of course), whether belief in the conclusion of an argument is warranted depends on what other arguments are possible. In particular, there can be arguments both for and against a proposition. These arguments can interfere with and sometimes defeat each other if, for example, there are good reasons to prefer one argument over another.

Definition 7 Two argument steps $\langle \Phi, p \rangle$ and $\langle \Psi, q \rangle$ CONFLICT if $KB \cup \{p, q\} \vdash \bot$ (*i.e.*, p and q are inconsistent).

In Pollock's system, this form of conflict is called *rebuttal*, and a second form, called *undercutting*, is also permitted. The intuition is that in rebutting conflict the value

of some proposition is at issue, while in undercutting conflict it is the applicability of a defeasible rule that is in doubt. In my approach, following Loui, I will impose certain criteria of defeat (below) which to some extent obviate the need for undercutting. Whether it is needed, either technically or because it is intuitive, remains an open question. If it is needed, I think there are some unresolved questions about the logical status of undercutting rules, quotation, and the like.

In any case, given two arguments that conflict, we are interested in preferring certain arguments over others. The following definition, adapted from Loui, makes this explicit:

Definition 8 An argument A is AT LEAST AS SPECIFIC as an argument B if there is a step $\langle \Phi, p \rangle \in A$ and a step $\langle \Psi, q \rangle \in B$ such that the steps conflict and $\Phi \vdash \Psi$. An argument is MORE SPECIFIC than another argument if it is at least as specific and the converse is not the case.

Clearly, there might be other grounds for preferring one argument over another besides specificity. Loui (1987) considers *evidence*, *directness*, and *preferred premise*, for example. As well, there might be domain-specific criteria, such as resource use or time limits in a planning context. I will have something more to say about this later when considering the application of the formalism to reasoning about plans, but determining a complete set of domain-independent preference criteria remains an open problem, and probably permanently so.

Finally, we can put these pieces together and propose the following:

Definition 9 An argument A DEFEATS an argument B if A conflicts with B and A is at least as specific as B.

Definition 10 An argument A is UNDEFEATED if there is no argument B that conflicts with A that is not itself defeated.

Note that for an argument A to be undefeated, there must be defeaters not just for the defeaters of A, but for any argument that conflicts with A. It would be possible to consider the weaker condition, but that allows potential defeaters to be ignored too easily.

Returning to our example, the two arguments

$$P(fred) \rightarrow \neg F(fred)$$

and

$$P(fred) \rightarrow B(fred) \rightarrow F(fred)$$

conflict at their last steps. The first argument is more specific than the second, since $KB \cup \{P(fred)\} \vdash B(fred)$ and not vice-versa. Therefore, the first argument defeats the second and is itself undefeated.

3.2.3 Justification and Warrant

Given these definitions leading up to the notion of an undefeated argument, the next thing to do would be to define a theory of *warrant* that describes what conclusions an ideal reasoner should draw from a given defeasible knowledge base. Since this ideal is usually unattainable in practice, if we are interested in implementing an argument-based reasoner we need to define a theory of *justified belief* or *justification* that describes what should be concluded given how much reasoning has been done and what has previously been concluded. The idea is that in the limit the justified conclusions converge to the warranted ones. Loui (1987) defines warrant (which he calls justification), but doesn't consider the process of constructing the arguments explicitly (although he does so elsewhere (Simari and Loui, 1992)). Pollock (1992a), defines warrant in terms of ultimately undefeated conclusions, and takes into account such phenomena as *collective defeat* and *self-defeat* (see also (Pollock, 1991)). He then considers the approximation process of justifying conclusions and proves the desired relationship to warrant.

I am not going to pursue these questions here. In the first place, it's not clear how much more can be said about warrant mathematically beyond the work cited above. There are some philosophical issues, and certainly debates about whether, in a particular case, some conclusion that was judged to be warranted should have been. But these discussions will ensue about any kind of defeasible reasoner and there's not necessarily a benefit to forcing the discussion into the rarified mathematical air of theories of warrant. Rather, I think we need to start applying the theories to domains of interest, such as reasoning about plans, and see whether they allow us to express our intuitions clearly and derive useful, reasonable conclusions. If some questions can be helped by carefully defining warrant and justification, then we can return and do so. But since in practice we will never be able to reason to anything remotely close to warranted conclusions, perhaps we should get on with the reasoning instead.

3.2.4 Discussion and Related Work

Before moving on to the application of argument systems to representing plans and reasoning about them, it is worthwhile to consider what makes them an attractive formalism for defeasible reasoning in general.

Konolige (1988) describes the application of a system of argumentation to reasoning about events in the Yale Shooting Problem. He refers to this form of reasoning as *direct inference*, since arguments are explicit objects of the theory, as opposed to model-theoretic approaches such as circumscription (McCarthy, 1980) or modal approaches such as default logic (Reiter, 1980) or autoepistemic logic (Moore, 1985). Konolige cites the following points in favour of direct over indirect inference (to each of which I have added some further comments):

1. Domain knowledge often includes information about the way defeasible arguments in that domain are related. This information can be represented straightforwardly in direct inference systems, whereas indirect approaches need to introduce new elements of the language whose only purpose is to encode these relationships.

I commented on this aspect of circumscription's unnaturalness, for example, in Chapter 2.

2. Direct inference systems are incremental. They are therefore ideally suited to resource-bounded, realistic reasoning, compared to the "all-or-nothing" global minimizations or fixed-point constructions of indirect systems. Loui (1991b) refers to this as *ampliativity*, and emphasizes that this type of nonmonotonicity in computation is different from and more important than the nonmonotonicity in evidence more commonly studied in work on nonmonotonic reasoning (see also (Loui, 1991a) in this regard).

3. The arguments produced by direct inference systems serve as an explanation for their conclusions. Konolige cites the usefulness of this for "debugging," but for my purposes the benefits are much greater. In mixed-initiative systems the arguments serve as the basis for conversation, a point that we will return to later when condisering related work on collaborative discourse (Section 3.5.3). Furthermore, if the conversation is about plans, the arguments correspond to the plans, in the sense that they include all the information needed to reason about the plan. The fact that the argument is an explicitly constructed, structured object is crucial to its use as a representation of plans.

4. The effects of chaining together arguments are available directly in a direct inference system. The point of this is presumably that indirect systems have trouble allowing a chain of plausible inferences to jointly turn out to be implausible (perhaps he's thinking of the Lottery Paradox), but I don't see this as such a significant point.

In the final analysis, the fact is that we are attempting to formalize plausible or intuitive forms of commonsense reasoning. Since the indirect methods do not represent the concepts involved in such reasoning (rules, defeat, reinstatement, *etc.*) naturally, "the results of trying to formalize defeasible reasoning within such a system are often surprising and counterintuitive." (Konolige, 1988) I made these same observations in Chapter 2 regarding the application of the indirect methods to the prediction problem. In this chapter I will concentrate on the fact that the arguments being constructed are explicit objects that I will use as the representation of plans. Also, the ampliative nature of argument-based reasoning is natural for interactive, dialogue-oriented reasoning.

Lin and Shoham (1989) introduce "argument systems" to serve as a uniform basis for nonmonotonic reasoning. They claim the approach is informally simple and intuitive, and formally show how the main nonmonotonic reasoning formalisms can be recast as special cases of their formalism. The definitions are in many respects identical to those presented above, although there are some subtle differences involving logical closure and completeness conditions. The emphasis is on the relationships between the other nonmonotonic formalisms as evidenced through the translation to argument systems, rather than on properties of the argument-based reasoning itself. In this sense the work is somewhat orthogonal to my motivations for using arguments. It does show the way in which the various indirect approaches only handle certain aspects of more general defeasible reasoning.

On a more informal note, there are several obvious similarities and differences with some of the nonmonotonic approaches. In the case of default logic, for example, the main theoretical problems in applying the framework have to do with the multiple extension problem, which makes it difficult to define a theory of warrant based on the default logic definitions. But default rules are intuitively very similar to the defeasible rules of arguments and in practice they are used the same way. Of course, there's no modal "derivability" operator in the argument-based approach. Rather, the definition of an undefeated argument takes into account the other arguments that are in force. An important difference between default logic and argumentation, however, is the third point listed above regarding arguments as explicit objects. In reasoning with default logic there is no analogue of the argument produced by argumentation. There are only the extensions of the theory, which correspond to the conclusions of arguments. It is not clear that default logic's fixed-point definition of extensions could be adapted to provide the sort of object we need to use arguments as a representation of plans (or anything else, for that matter). Interestingly, Doyle's TMS (Doyle, 1980b,a), although subsequently co-opted by the nonmonotonic reasoning community as "simply" an implementation of default logic (or other formalisms), is in fact much closer in spirit to the type of dialectical argument-based defeasible reasoning described here.

The other close cousin of argument systems among the nonmonotonic reasoning approaches is the theory formation approach to defeasible reasoning (Poole et al., 1987; Poole, 1988). Like my argument-based approach, the emphasis is not on model-theoretic characterizations of desirable conclusions, but rather on the process of finding such conclusions. Drawing defeasible conclusions is seen as abductively forming a scientific theory, based on both non-defeasible knowledge and classes of hypotheses that can be assumed. Thus is it similar to an argument system where the defeasible rules have no premises. The problem is that by not representing the rules explicitly, certain types of knowledge about interactions between rules cannot be expressed. More importantly for my purposes, the resulting theories are "flat" sets of assumptions. In the argument-based approach, the resulting "theory" (argument) has an internal structure that, I claim, in the case of arguing about plans, corresponds to plans. I have already noted that such structure is necessary for explaining conclusions, and it is otherwise useful, for example in expressing preferences among rules and arguments. But argument systems and theory formation certainly share the motivation that we need to look at the *process* of defeasible reasoning and not just at its conclusions if we are going to adequately model commonsense reasoning.

Finally, I want to address briefly the question of whether what we're doing when we're reasoning defeasibly isn't just some form of probabilistic or statistical reasoning. If that is the case, perhaps we would be better off using (apparently) well-understood formalisms from probability and decision theory rather than developing new ones. This question, of course, has a long history in AI and I am not going to rehash it all here. The discussion is excellently summarized in two special issues of *Computational Intelligence*, with position papers by Cheeseman (1988) and Kyburg (1994) respectively. One way of looking at the defeasible rules employed in argumentation is as statements of qualitative conditional probability. This yields the interpretation of defaults like "birds fly" as

"most things that are birds are things that fly." However, another interpretation of defeasible rules is that they represent conventions or conversational norms. In this view, "birds fly" means that although there are certainly exceptional birds that don't fly, if the bird under discussion was one of these I would (or should) have mentioned it. This interpretation is not at first glance statistical (although it could be argued that conventions that did not conform to the probabilities would be inefficient ones). It seems that in some, if not all, cases the probabilists might be right so perhaps we should listen to them.

However, on one important point decision theory is silent. This is the construction of the model that we wish to evaulate—the states of the world in decision theory. Typically there are vast numbers of states arising from even a simple propositional description as used in AI, and we can't expect to be able to assign utilities to all of them. This is the problem of so-called "small worlds." Therefore an essential question that the probabilists need to answer is how they form a model that they can subsequently evaluate. One approach is via a notion of acceptance: certain propositions are accepted with probability 1, at least for the present reasoning. Kyburg (1990) discusses this approach and compares it to work on nonmonotonic reasoning since many of the same issues regarding formalizing our intuitions arise. Argumentation offers another possibility, since in a sense the rules used in arguments and their conclusions are "accepted," depending on what else we know. Loui (1990b; 1990a) describes how this view can be used in conjunction with numerical utilities and compares it to strict Bayesian decision theory in some detail. In particular, he considers planning with utilities and discusses the way traditional AI planning and decision theory differ in their characterization of states of the world and the desirability of those states. He writes that "the next question to be studied is how to interpret the planning ideas of goal-directed search" (Loui, 1990b) and offers some suggestions. I believe that the view of planning based on argumentation presented in the rest of this chapter also addresses part of this question. Among other things, it shows how the algorithms developed for AI planners can be used in argumentation, at least in some cases. It also emphasizes that there is more to argumentation than heuristic search, especially in the context of mixed-initiative systems. I will return to the question of integrating statistical information with argumentation in Section 3.6.2.

3.3 Arguing About Plans

With that formal basis, I can now turn to the issue of using arguments to reason about actions and plans. The approach will be to construct a set Δ of defeasible rules that represents our knowledge of causality (among other things). These can then be used to construct arguments that a certain course of action under certain explicit conditions will achieve certain goals. These arguments can be evaluated relative to a knowledge base KB that includes both general (non-defeasible) "laws" and a specification of the situation under consideration (the "initial" conditions; although everything is temporally qualified, these are the starting points for reasoning). I claim that such arguments

provide precisely the representation of plans needed for mixed-initiative planning and other complicated plan reasoning.

The starting point for argument-based reasoning is the set of defeasible rules Δ and the non-defeasible knowledge base KB. Within Δ there are two classes of rules that we need to consider: defeasible causal rules and permitted assumptions (*i.e.*, rules with no premises). In both cases we need to ensure that Δ contains a sufficiently broad set of rules to allow as many plausible lines of reasoning (*i.e.*, plans) to be constructed as may be encountered in the various plan reasoning tasks in which we are interested. This emphasis on breadth is a major theme of my approach.

I start, therefore, with an exposition of formulating defeasible causal rules and consider the qualification problem in that context. I then consider an approach to the frame problem that combines the explanation closure technique from the previous chapter with assumptions about non-occurrence of relevant events. I then consider the use of the defeasible rules in constructing plans-as-arguments, and develop several criteria with which to evaluate and compare them. To provide a connection to previous planning formalisms, I then reconstruct one of them (the SNLP formalism) within the plans-as-arguments framework. This analysis reveals both the important aspects of traditional planning and the relationship to more general defeasible reasoning. As stated at the outset, the development of the plans-as-arguments approach is by no means complete. The chapter therefore concludes with discussions of related work on plan representation and reasoning, some of the many issues for future work, and finally a summary of the contributions of this part of the dissertation.

3.3.1 Defeasible Causal Rules

My defeasible rules about causality start from the causal ETRY axioms introduced in Section 2.2.4. Recall that the form of these axioms was:

$$\forall t \,.\, p_1 \wedge \cdots \wedge p_n \wedge Try(\pi, t) \supset \exists e \,.\, \textsc{Event}(e) \wedge \phi,$$

where π is a program (action), EVENT an event-type predicate, and $p_1, \ldots, p_n$ the sufficient conditions for execution of the action to bring about the occurrence of such an event. In Section 2.2.4 I emphasized the necessity of separating action attempts from the events they cause, and I will not repeat that argument here, although it still applies.

In the context of defeasible reasoning, what is troubling about the ETRY axioms is that they run into the *qualification problem*—no matter how many preconditions we list, there are always other conditions that might prevent an appropriate event from occurring. Similarly, although not usually considered in terms of the qualification problem, not all preconditions of an action are necessary, or at least we may not need to know or ensure their truth for the action to succeed (*i.e.*, cause an event of the appropriate type). For example, if the preconditions for filling a boxcar from a hopper are that the hopper be full and the boxcar be under the hopper, we might try filling the boxcar without checking the hopper. Perhaps the hoppers are usually full, or perhaps we don't have time to check the hopper. Nonetheless, if the hopper was full, the boxcar

will be filled, so this is at least a plausible thing to try, if not the "best" thing to do in some sense.

As a consequence, I see the qualification problem as a sword that cuts both ways through the notion of "precondition" as it has been used in AI planning. This is not to say that we don't have legitimate intuitions about certain conditions being necessary, or at least favorable, to the successful execution of actions. It does mean that we need a weaker way to express these intuitions than that afforded by traditional planning preconditions. My defeasible causal rules are based on these intuitions, and treat preconditions as "influences," either positive or negative or both.

The basic defeasible rule for an action π whose execution leads to occurrence of an event of type EVENT is:

$$Try(\pi, t) \rightarrow \exists e \,.\, \textsc{Event}(e) \wedge \phi.$$

The temporal variable t is schematic, and ϕ represents additional constraints on the event e, such as role bindings and temporal constraints. A simple example involving starting a car is:

$$Try(turnkey, t) \rightarrow \exists e \,.\, \textsc{RunEngine}(e) \wedge t : time(e). \tag{3.1}$$

That is, turning the key starts the engine running (see Section 2.4.1 for why this should be treated as an event). The definition of the RUNENGINE event (EDEF axioms), including it's effects, remains non-defeasible (*i.e.*, in KB). This is because of my treatment of events as cognitive objects that organize patterns of change (*cf.* Section 2.2.3).

Now, one intuitive precondition for this action-event relationship is that the car's gas tank cannot be empty. There is therefore another rule:

$$\neg TankEmpty(t), Try(turnkey, t) \rightarrow \exists e \,.\, \textsc{RunEngine}(e) \ldots \tag{3.2}$$

However, it's also the case that if the tank is empty the car won't start (let's assume this is defeasible—there might be a backup tank or the gauge might be inaccurate). That is, there is also a rule like:

$$TankEmpty(t), Try(turnkey, t) \rightarrow \neg\exists e \,.\, \textsc{RunEngine}(e) \ldots \tag{3.3}$$

Note that we still keep the original rule (3.1) in Δ since it represents a plausible line of reasoning or course of action. We are adding additional *more specific* rules to Δ that may apply in different situations. In this case (3.3) conflicts with both (3.1) and (3.2). It is more specific than (3.1) but neither more nor less specific than (3.2).[2]

This type of precondition, which has both positive and negative rules associated with it, I refer to as as a *hard precondition*. Most of the discussion of preconditions in

[2]Another possibility would be to make the negative precondition rules undercutting defeaters (in Pollock's terms) of the basic rule (3.1) rather than rebutting defeaters. There are complications, since it would also have to undercut (3.2) and we would need to make sure the specificity was properly handled. Since (3.3) is defeasible, it doesn't say that the event can't occur, only that, if this is all that's going on, it won't. My intuitions vary on this, and it may be that we need both kinds of rules to properly represent all types of preconditions. But for concreteness, I will use the rebutting form in what follows.

AI is about hard preconditions, but not all preconditions need be hard. Note that if defeasible rules contraposed, they would all be hard, but defeasible rules don't contrapose. Preconditions that only have positive rules (*i.e.*, that can only strengthen an argument for an action's success) I refer to as *soft preconditions*. An example might be that if it is not raining, the car is likely to start, but even if it is raining, it's not sufficiently likely not to start to support an argument to that effect. Finally, for completeness, preconditions that can only defeat arguments about an action's success I will refer to as *exceptional preconditions*. An example of one of these might be the classic "potato in the tailpipe:" having a potato in the tailpipe is sure to prevent the car from starting, but knowing one is not there does not improve the chances of its starting relative to not knowing either way.

Finally then, a defeasible causal rule

$$p_1, \ldots, p_n, Try(\pi, t) \rightarrow \exists e \,.\, \textsc{Event}(e)$$

generates a lattice of rules relating π and *Event* and ordered by specificity like that shown in Figure 3.3. The idea is that each of these is at least a plausible line of reasoning, however incomplete and subject to counter-argument using the more specific rules. Similarly, from hard and exceptional preconditions we get a lattice of negative rules that state that falsity of one or more of the preconditions is reason to believe that the action will not succeed. There are many possibilities. Whether the bottom rule of the negative lattice (*i.e.*, $Try(\pi, t) \rightarrow \neg\exists e.\textsc{Event}(e)$) is added depends on whether trying the action "blind" is more or less likely to succeed. If both positive and negative bottom rules are present in Δ, then we can't argue convincingly either way, which might be the intuitive thing.

The exact structure of these lattices, in particular the negative rules, may depend on our knowledge of how preconditions interact. For example, we might feel that a particular precondition was a negative influence unless all the others were true, in which case the action was likely to succeed. My position on this is like the position on the explanation closure technique secribed in Section 2.3.2. That is, each of the defeasible rules has epistemic content—it represents knowledge of the domain. In some cases, the generation of the lattices from a STRIPS-like description of preconditions can probably be automated (as with the generation of explanation closure axioms in simple domains). But in general we will need to consider exactly how each of what we are calling "preconditions" affects our reasoning. This is the price we pay for moving from artificial, overly-simplified action descriptions to more natural and intuitive ones.

3.3.2 Explanation Closure Assumptions

Another aspect of causal reasoning and reasoning about actions and plans arises in the context of the frame problem. Recall from Section 2.3.2 that this is the problem of specifying what doesn't change as a result of executing an action. As I noted there, this is sometimes referred to as persistence reasoning, the intuition being that properties "persist" unless changed by an action or event. In Chapter 2, I introduced the idea of *explanation closure* as an alternative characterization of this type of reasoning.

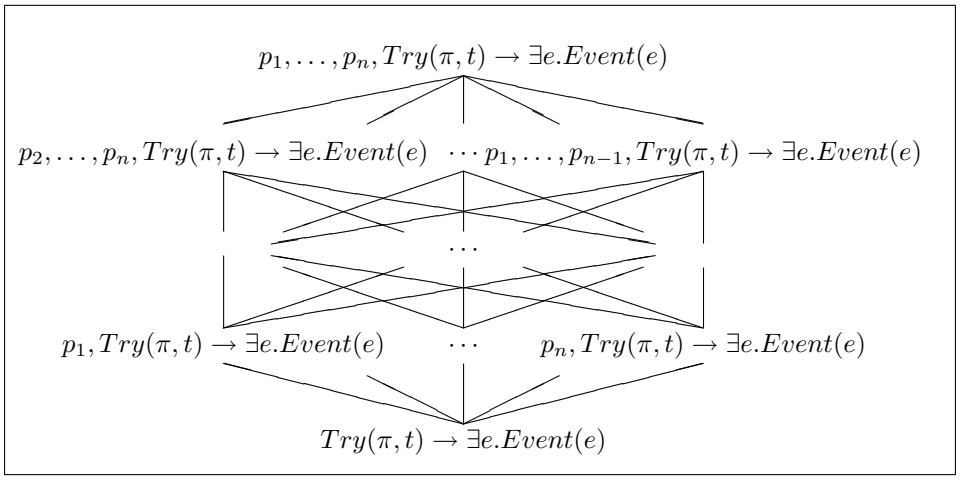


Figure 3.3: Positive precondition rule lattice

Recall that the idea was to provide axioms that describe completely under what conditions a property can change. Reasoning about non-change then reduces to reasoning about what actions occurred (or didn't occur). I commented that one of the attractive features of the explanation closure approach was that it made the assumptions underlying the "persistence" reasoning explicit and available for reasoning about, rather than embedding them in the model theory.

My approach will be to use the explanation closure axioms described in Section 2.3.2, and then permit plans to include assumptions related to them. Recall that the basic form of the explanation closure axiom for a property P was

$$\forall t, t' \,.\, P(t) \wedge \neg P(t') \wedge t : t' \supset \exists e \,.\, (E_1(e) \vee E_2(e) \vee \cdots) \wedge time(e) : t'.$$

where the E_i are the event type predicates that (possibly) affect the truth of P. In order to reason about persistence, we need something like the contrapositive of this, such as

$$\begin{aligned} &\forall t_1, t_2 \,.\, P(t_1) \wedge \\ &\quad \neg[\exists e \,.\, (E_1(e) \vee E_2(e) \vee \cdots) \wedge EndsAfter(time(e), t_1) \wedge time(e) \prec\!: t_2] \\ &\qquad \supset P(t_2), \end{aligned}$$

where

$$EndsAfter(t, t') \equiv t' \prec t \vee t' : t \vee Overlaps(t', t) \vee Starts(t', t) \vee t' \sqsubset t.$$

This persistence axiom follows from the EXCP axiom and the TRPT theorem of the interval temporal logic. Although it seems unwieldy, it is quite intuitive. It states that if a property holds at time t_1, and no event of a type that affects P occurs over a time that ends after t_1 and before t_2, then P will still hold at t_2. The temporal condition is illustrated in Figure 3.4. Note that is is tempting to use the condition

$$t_1 \prec\!: time(e) \prec\!: t_2$$

to capture the idea of e interfering with P between t_1 and t_2. When events are not instantaneous, however, that condition is too weak since e can be "ongoing" over t_1. To see this, suppose event e causes $\neg P$ immediately after $time(e)$, $t_1 \sqsubset time(e)$, and $time(e) \prec t_2$. Then the weaker condition would allow us to prove persistence of P from t_1 to t_2 despite the fact that e, once it finishes, will "clobber" P.[3]

If we introduce the following abbreviation for the occurrence formula:

$$NC(P, t_1, t_2) \equiv \neg[\exists e \,.\, (E_1(e) \vee E_2(e) \vee \cdots) \wedge EndsAfter(time(e), t_1) \wedge time(e) \prec\!: t_2],$$

then assuming the EXCP axioms are in KB, we have that

$$KB \cup \{P(t_1), NC(P, t_1, t_2)\} \vdash P(t_2).$$

This is exactly the condition we need to reason about persistence.

[3]The *EndsAfter* relation is appropriate when the effect starts at the end of the event. In general, of course, the temporal relationship will depend on the temporal structure of the event and its effects.

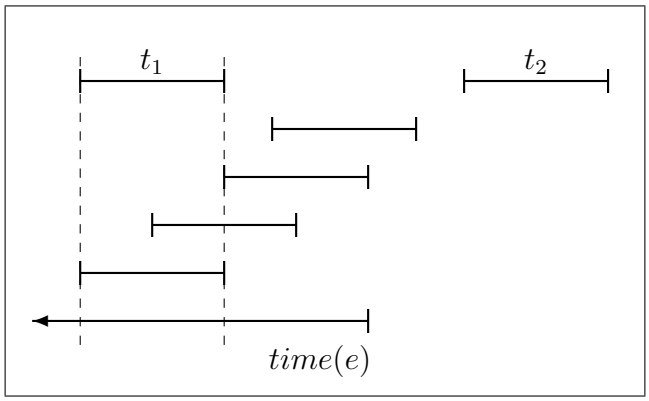


Figure 3.4: Temporal relations in the peristence axiom

For reasoning defeasibly about plans, I will allow the agent to *assume* the NC condition and use it to reason about persistence, subject to other constraints on the plan (argument). We therefore add the following assumption to Δ:

$$\rightarrow NC(P, t_1, t_2)$$

for any property P and times t_1 and t_2. An example of how this is used will be presented shortly, but the important thing, all the notation aside, is that I am using explanation closure in conjunction with assumptions about non-occurrence of events to reason about persistence. It is worth emphasizing that although this looks like "just" a persistence assumption, especially in its abbreviated form, the implications of assuming $NC(P, t_1, t_2)$ are both deeper and clearer. Such an assumption places strong constraints on what events can or cannot occur over the appropriate interval, as we desire. These semantics of the NC assumption are given directly in the temporal logic axioms, rather than being defined in a nonmontonic semantic theory (*e.g.*, (Shoham, 1988)) or via a procedural technique (*e.g.*, TMM (Dean and McDermott, 1987)).

3.3.3 Formal Properties of Plans as Arguments

We are now ready to define various properties of plans-as-arguments based on the defeasible causal rules and consider the relationships between them. In several cases these definitions correspond to various properties of plans considered previously in the literature on the formal foundations of planning. In this case, however, they arise as intuitive and natural properties of arguments, thereby broadening their scope beyond simple planning formalisms. In the next section, however, I will return and reconstruct the traditional model of planning within the context of defeasible reasoning.

I want to emphasize that my purpose in using the argument systems formalism is not to provide a precise mathematical foundation for planning. I don't think that is possible in any meaningful way for the kind of realistic, mixed-initiative situations we are interested in. Rather, I want representations that lend themselves to the kind of reasoning that we observe in our dialogues. There are no theorems here, since what is at issue is the relationship between the formal system and our intuitions. The hope is that the formal definitions at least make it clear whether the two agree or disagree.

My first definition is a very simple one, so much so that most previous approaches to defeasible reasoning presuppose it as part of the definition of an argument.

Definition 11 An argument A is INCONSISTENT if $KB \cup Props(A) \vdash \bot$, otherwise it is CONSISTENT.

As with well-formedness, we do not want to rule out inconsistent arguments *a priori*. Rather, we want to be able to reason about them, about why they're inconsistent, and so on. Again, since most previous work on defeasible reasoning has concentrated on a single agent drawing conclusions, arguments have been assumed to be consistent. While we will generally want to construct consistent arguments (resources permitting), we may need to recognize or communicate about inconsistent ones. Note that allowing

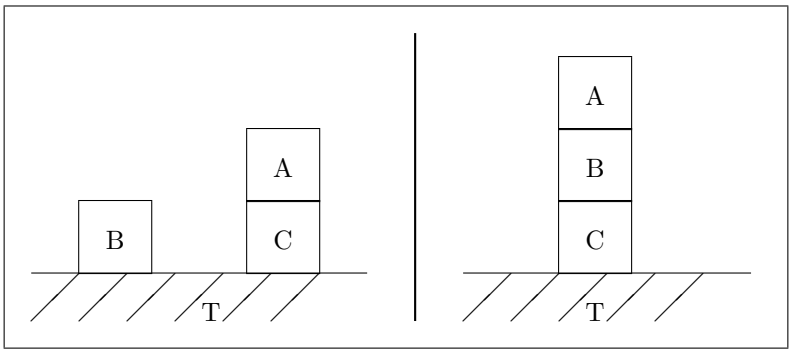


Figure 3.5: The Sussman Anomaly

inconsistent arguments affects the definition of defeat, since an inconsistent argument is at least as specific as any other argument according to Definition 8. This may or may not be what we want, but the issues are separable and we could change the definition of specificity if needed.

Since arguments needn't be "complete" or "maximal" in any way, consistency is not a very strong condition. For example, consider the Sussman anomaly problem diagrammed in Figure 3.5, where the goal is to transform the left-hand scenario into the right-hand one by stacking blocks. This situation is described by the following facts in KB, where I am using a simplified integer model of time to simplify the example:

$$Clear(A,0), Clear(B,0), Clear(T,0), On(A,C,0)$$

We then have the definition of the stacking event, also in KB:

$$\forall e,t,x,y \,.\, \textsc{Stack}(e,t,x,y) \supset On(x,y,t+1)$$
$$\forall x,t \,.\, Clear(x,t) \equiv [x = T \vee \neg\exists y \,.\, On(y,x,t)]\}$$

This is a simplification of the definition of Stack in Chapter 2, and states that after stacking x on y, x is on y, and hence that y is not clear (unless y is the table, T, which is always clear). We also have the defeasible causal rules (*cf.* Section 3.3.1) for attempting to stack two blocks, in Δ:

$$Try(stack(x,y),t) \rightarrow \exists e \,.\, \textsc{Stack}(e,t,x,y)$$
$$Clear(x,t), Clear(y,t), Try(stack(x,y),t) \rightarrow \exists e \,.\, \textsc{Stack}(e,t,x,y)$$
$$\neg Clear(x,t), Try(stack(x,y),t) \rightarrow \neg\exists e \,.\, \textsc{Stack}(e,t,x,y)$$
$$\neg Clear(y,t), Try(stack(x,y),t) \rightarrow \neg\exists e \,.\, \textsc{Stack}(e,t,x,y)$$
$$\rightarrow Try(stack(x,y),t)$$

With these rules, the blocks being clear are both hard preconditions.

From these, we can construct the well-formed plan shown in Figure 3.6. Ignore the exact definition of the goal G for now and assume that the final step of the argument is deductively justified. The point is that the plan in Figure 3.6 is consistent even though

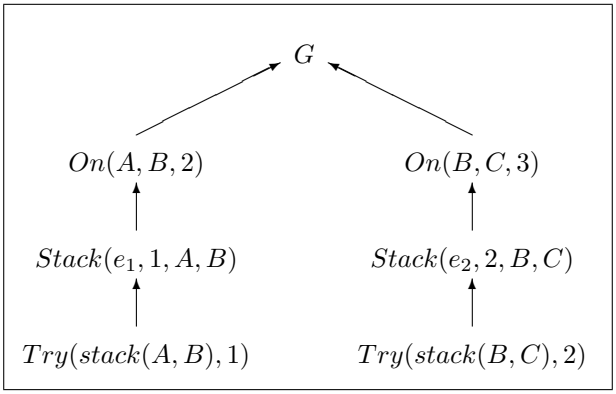


Figure 3.6: Initial plan for the Sussman Anomaly

it is obviously incorrect (stacking A on B first means that attempting to stack B on C will fail). But it's important that it is a plan—for example, it might be a "first-cut" plan produced before considering interactions, or it might be a plan being described by a person as they work on a solution to the problem. To make sense of why it is a "bad" plan, we need some more definitions.

Definition 12 An argument A' is a SPECIALIZATION of an argument A if there exist steps $\langle \Phi, p \rangle \in A$ and $\langle \Phi', p \rangle \in A'$ such that $KB \cup \Phi' \vdash \Phi$ but $KB \cup \Phi \nvdash \Phi'$.

That is, some step in the more specialized plan uses a more specific defeasible rule to reach the same conclusion. Note that the specialization relation is not anti-symmetric and thus does not define an order on arguments. That is, for two arguments A and A', we can have that A is a specialization of A' and also that A' is a specialization of A, at different steps, of course. I think this reasonable, since such situations arise naturally, for example when two agents are refining different parts of their joint plan. For any given step in an argument, however, the specializations of that step are ordered by specificity (as in the lattice of precondition rules, Figure 3.3).

The definition of specialization is used to express the following property of what might be considered "partial" arguments:

Definition 13 An argument A is VALID if (a) it is consistent; and (b) there is no specialization of it that is inconsistent.

Thus a valid argument is one in which we haven't simply ignored inconsistency.

The notion of valid argument makes sense of the Sussman example, since Figure 3.7 shows a plan that is a specialization of that shown in Figure 3.6 and that is inconsistent (since $Clear(B, 2)$ and $On(A, B, 2)$ are inconsistent given KB). The intuitively correct plan, obviously, is doing the actions $stack(A, T)$, $stack(B, C)$ and $stack(A, B)$ in that order. One valid argument that makes this clear is shown in Figure 3.8. Note that the step from e_1 occurring to C being clear is sanctioned by KB, so this argument is well-formed.

So the the concept of a valid, well-formed plan is one in which there aren't obvious contradictions or omissions. In a sense, these are "internal" properties of the plan, concerned with the rules used in its construction. The next step is to consider the assumptions on which the plan is based and their relation to the defeasible knowledge base. My next definition partitions the premises of an argument into two sets:

Definition 14 For any argument A and defeasible knowledge base $\langle KB, \Delta \rangle$, $Premises(A)$ can be partitioned into two disjoint, exhaustive sets, the ASSUMPTIONS and the CONDITIONS:

$$\begin{aligned} Assumptions(A) &= \{q | \langle \emptyset, q \rangle \in \Delta\} \\ Conditions(A) &= Premises(A) / Assumptions(A) \end{aligned}$$

That is, the assumptions are those premises sanctioned by a premise-free defeasible rule, and the conditions are all other premises.

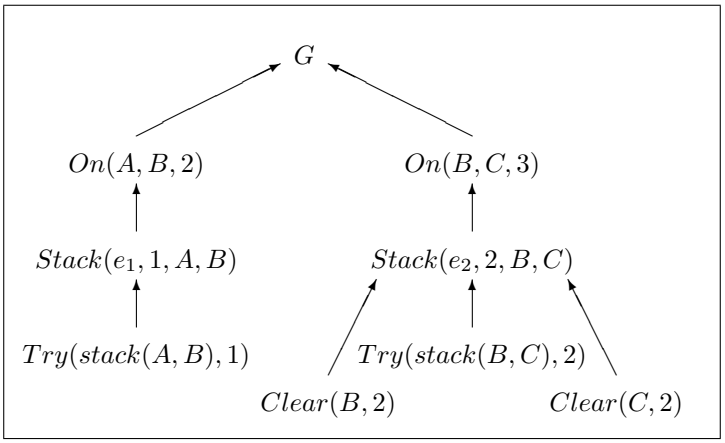


Figure 3.7: An inconsistent specialization of the Sussman plan

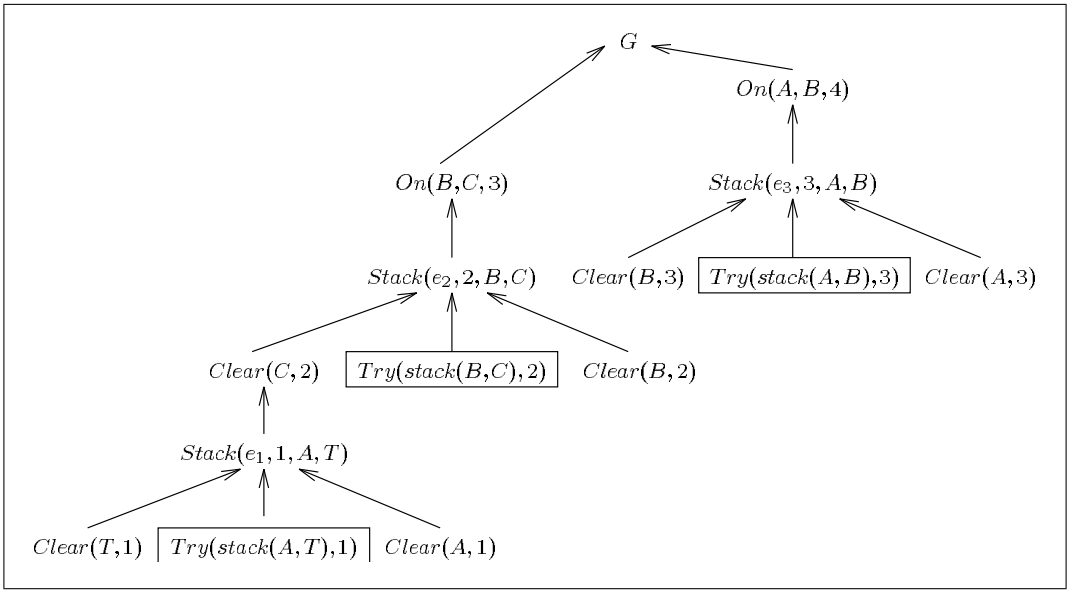


Figure 3.8: A valid plan for the Sussman Anomaly

Definition 15 An argument A is SUPPORTED relative to $\langle KB, \Delta \rangle$ if $KB \vdash Conditions(A)$.

Thus the supported plans are those for which the non-defeasible knowledge sanctions the conditions on which the plan depends (and the defeasible knowledge sanctions its assumptions). The plan shown in Figure 3.8 is not supported, since the knowledge base only includes information about the locations of the blocks at time 0, not at the times required by the conditions of the plan. This is, of course, the frame problem, but in general plans can be unsupported for other reasons, such as not having had time to discharge an assumption or simply making an incorrect assumption.

If we add the explanation closure-based persistence assumptions described in Section 3.3.2, we can then form a supported plan for the Sussman problem. That is, we add

$$\rightarrow NC(Clear(x), t_1, t_2),$$

to Δ and add the NC definition for the property $Clear$, for any block x, to KB:

$$NC(Clear(x), t_1, t_2) \equiv \neg[\exists e, t, y \,.\, \textsc{Stack}(e, t, y, x) \wedge EndsAfter(t, t_1) \wedge t \prec: t_2],$$

together with the EXCP axiom for $Clear(x)$:

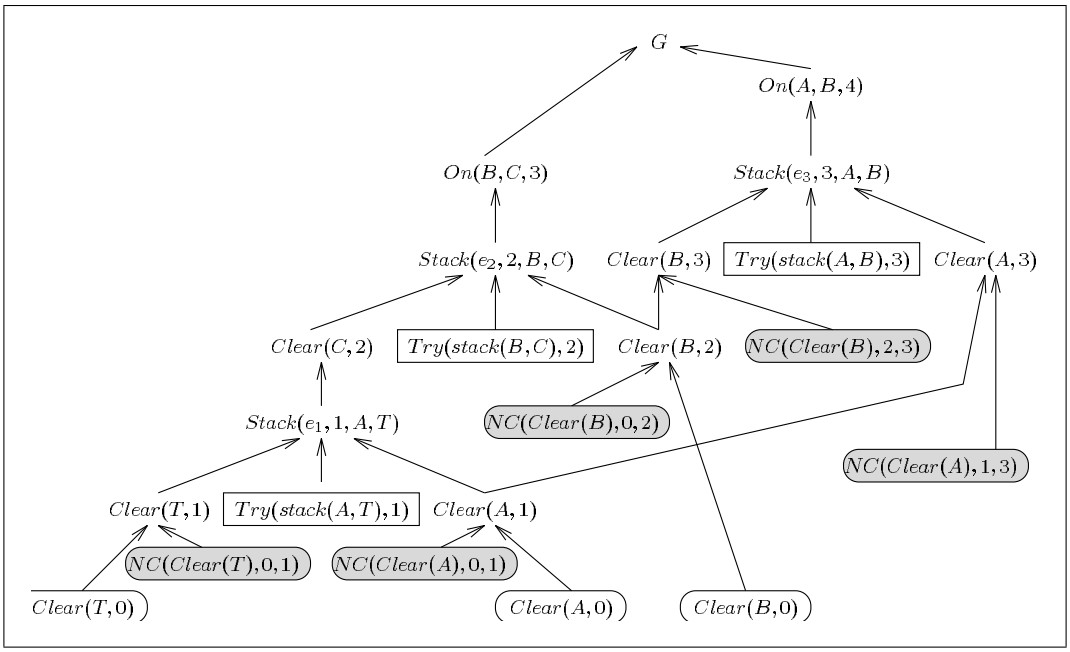
$$\forall x, t_1, t_2 \,.\, Clear(x, t_1) \wedge \neg Clear(x, t_2) \wedge t_1 : t_2 \supset \exists e, t, y \,.\, \textsc{Stack}(e, t, y, x) \wedge t : t_2.$$

With these additions, the plan shown in Figure 3.9 is a supported, valid, well-formed plan for the Sussman problem. The actions in the plan are boxed, the persistence assumptions are shaded, and the other premises are all in the description of the initial conditions. If we expand the definitions of the NC conditions, we can verify that the plan is indeed consistent.

With this final example, which illustrates the representation of plans as arguments, the defeasible causal rules, and the explanation closure-based persistence assumptions, I am going to conclude the consideration of the formal properties of plans as arguments. In part this is because I feel that further formalization in the abstract would not contribute to the goal of developing a representation suitable for the real-world problems posed by mixed-initiative planning. That is, the properties I have presented are interesting because they correspond quite closely to aspects of traditional formal models of planning. This point will be emphasized in the next section, where I present a reconstruction of the SNLP model of STRIPS planning within the defeasible reasoning paradigm. But beyond that, I don't think that our intuitions are sufficiently clear to warrant immediate formalization. Rather, we need to apply the formalism in domains of interest (such as the TRAINS domain plan reasoner to be described in Chapter 4) and then revise the formal model to account for those aspects of the phenomena that are sufficiently well understood. Section 3.6 presents some of the issues that I expect will need to be covered in such future work.

3.4 Reconstructing Traditional Planning

I have argued throughout this dissertation that we need a broader view of plans than that afforded by traditional models of planning, if we are to build systems that can



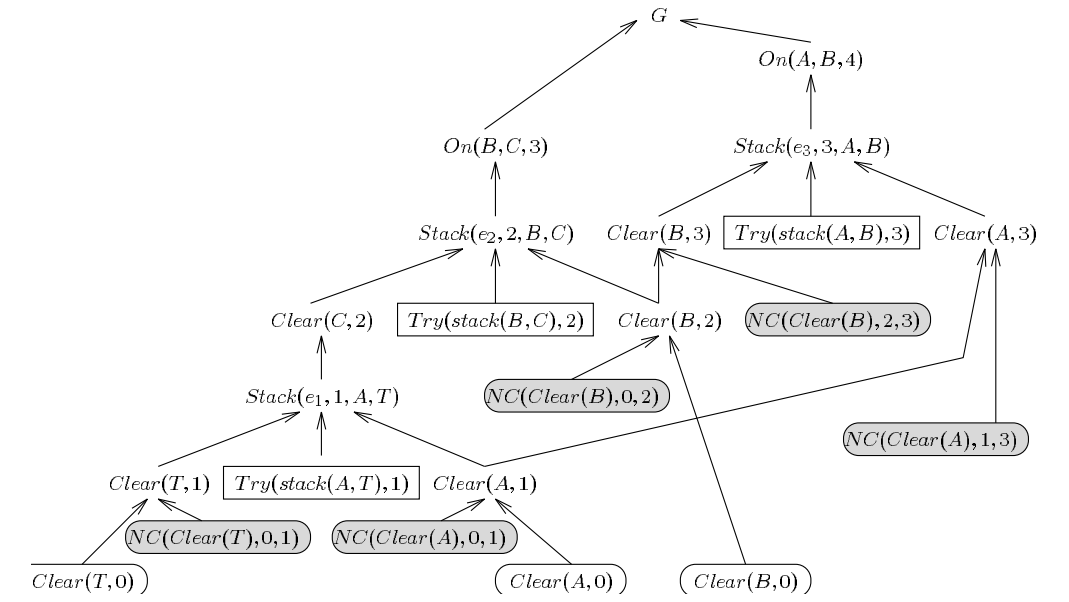


Figure 3.9: A supported plan for the Sussman problem

participate in mixed-initiative planning scenarios such as that described in Section 3.1. Nonetheless, we cannot simply ignore the extensive and important previous work on AI planning. This section investigates the relationship between traditional STRIPS-style planning and the plans-as-arguments approach that I am advocating by reconstructing the former within the defeasible reasoning paradigm. I feel that both sides benefit from the reconstruction. The assumptions underlying traditional planning are separated from the algorithmic and implementation details of the model and are considered in the context of a more general assumption-based reasoning system which, I have argued, is something we need to do for mixed-initiative planning. The defeasible reasoning system benefits from being applied to a problem of fundamental importance to AI, rather than the more abstract "birds fly" problems more commonly used to illustrate them.

This section proceeds as follows. First, I will review the definitions of the SNLP model of planning (McAllester and Rosenblitt, 1991). This model marked the beginning of a rekindling of interest in the formal foundations of STRIPS planning. It has since led to work such as the UCPOP model (Penberthy and Weld, 1992) and the BURIDAN planner (Kushmerick et al., 1994). I then reconstruct the SNLP model as a defeasible knowledge base with facts and rules and show how SNLP plans correspond to arguments satisfying certain criteria. Finally, I summarize what I think is gained from this admittedly somewhat artifical, technical exercise.

3.4.1 The SNLP Formalism

The following definitions are taken from (McAllester and Rosenblitt, 1991).

Definition 16 An SNLP OPERATOR is a tuple $\langle o, P, A, D\rangle$ where o is the OPERATOR NAME, P a set of PREREQUISITES, A a set of propositions ADDED by the operator, and D a set of propositions DELETED by the operator.

Definition 17 An SNLP STEP is a tuple $\langle O, n\rangle$, where O is an operator and n is the INDEX of the step.

That is, steps are instances of operators, since the same operator can be used several times in a plan. The indexes serve only to distinguish instances and do not imply any temporal ordering. In what follows, I will ignore the mapping between steps and operators (McAllester and Rosenblitt's *symbol table*), and assume that I can refer to the preconditions of a step, for example, and mean the preconditions of the operator of the step.

Definition 18 An SNLP CAUSAL LINK is a tuple $\langle s, P, w\rangle$, written $s \xrightarrow{P} w$, where s and w are steps and P is a proposition. The meaning of this is that (a) s is before w, (b) s adds P, and (c) w requires P.

Note that the condition that "no step between s and w adds or deletes P" is not a part of the definition of a causal link. Rather, it is enforced by subsequent constraints on acceptable plans. It might therefore be more accurate to call these "potential" or "intended" or "*ceteris paribus*" causal links."

Definition 19 An SNLP operator is a THREAT to a casual link $s \xrightarrow{P} w$ if it adds or deletes P.

In subsequent discussion, McAllester and Rosenblitt state that considering operators that add P to be threats is only necessary for systematicity, and that in practice only operators that delete P need to be considered (as in NONLIN (Tate, 1977)). Again, I will ignore the mapping from steps to operators and talk about a step being a threat if its operator is a threat.

Definition 20 An SNLP PLAN is a tuple $\langle S, L, C \rangle$, where S is a set of steps, L is a set of causal links, and C is a set of ordering constraints "$w < v$" on steps.

The goals of a plan are expressed as the prerequisites of a dummy step *FINISH*. The initial conditions are expressed as the add-list of a dummy step *START*. Any plan must include both dummy steps in S.

Definition 21 An SNLP plan $\langle S, L, C \rangle$ is COMPLETE if:

1. For every step w in S, if w requires P, then there is a link $s \xrightarrow{P} w$ in L for some s in S.

2. For every step v in S, if v is a threat to some link $s \xrightarrow{P} w$ in L, then either $v < s$ or $v > w$ is in C. These constraints are called SAFETY CONDITIONS.

The first clause ensures that all preconditions (including the goals, expressed as prerequisites of *FINISH*) are satisfied. The second clause ensures that "no step between s and w adds or deletes P."

It remains to ensure that the ordering constraints are consistent:

Definition 22 A TOPOLOGICAL SORT of an SNLP plan $\langle S, L, C \rangle$ is a linear ordering of S such that:

1. *START* is first in the ordering and *FINISH* is last.

2. For each causal link $s \xrightarrow{P} w$ in L, s precedes w in the ordering.

3. For each ordering constraint $s < w$ in C, s precedes w in the ordering.

Any topological sort of a plan is a solution to the corresponding planning problem, and there is no solution if there is no topological sort (*i.e.*, the ordering constraints in C and implied by L are inconsistent).

3.4.2 Defeasible Reasoning and SNLP Planning

In order to reconstruct the SNLP model within our defeasible reasoning system, we first need to construct a defeasible knowledge base $\langle KB, \Delta \rangle$ from which to construct arguments (plans). The operator definitions are translated into defeasible rules. Threats to and safety conditions on causal links are expressed using additional rules, and asumptions are permitted regarding causal links and operator orderings. Plans are then arguments from the initial conditions to the goals that satisfy certain conditions.

Since SNLP is capturing STRIPS planning, the explicit temporal model introduced in Chapter 2 and used in Section 3.3 is not appropriate. Rather, temporal reasoning is implicit and represented relative to steps in the plan. In order to remove some of the implicitly procedural aspects of the SNLP model, however, I will use a somewhat more explicit representation for steps and propositions than that used by SNLP itself. Steps will be terms of the language, and I again assume a mapping from steps to operators. The predicate $Occurs(s)$ means that the operator of step s is executed. Propositions can either be stated to hold immediately prior to a step's execution, written $P(in(s))$, or immediately after, written $P(out(s))$. Actions, of course, are instantaneous and sequential in the STRIPS model

Given an SNLP operator $\langle o, P, A, D \rangle$ as in Definition 16, we represent the preconditions using a defeasible *precondition rule*:

$$P_1(in(s)), \ldots, P_n(in(s)) \rightarrow Occurs(s).$$

Viewing steps as something like events, the add- and delete-lists can be specified using non-defeasible *add* and *delete* rules:

$$Occurs(s) \supset A_i(out(s))$$
$$Occurs(s) \supset \neg D_i(out(s))$$

The important issue is how to formalize causal links, threats, and the resolution of threats. A causal link is part of the following defeasible rule, for any property P and steps s and w:

$$s < w, P(out(s)), CLINK(s, P, w) \rightarrow P(in(w)) \tag{3.4}$$

That is, following Definition 18, s is before w, s adds P, and w requires P. As in SNLP, this is the formalization of the STRIPS assumption, and also as in SNLP, the resolution of threats relies on a global evaluation of the plan. The planner is allowed to assume causal links, and threats conflict with these assumptions unless they are resolved by safety conditions. We get the following set of defeasible rules:

Basic rule:	$\rightarrow CLINK(s, P, w)$
Threat rule:	$\neg P(out(v)) \rightarrow \neg CLINK(s, P, w)$
Demotion:	$\neg P(out(v)), v < s \rightarrow CLINK(s, P, w)$
Promotion:	$\neg P(out(v)), v > w \rightarrow CLINK(s, P, w)$

These rules are interpreted as follows. The planner can assume causal links. Arguments based on these assumptions can be defeated by an argument that uses a step that

deletes the condition (or adds it if it's a negative condition). This defeater can itself be defeated by the more specific rule that uses the safety conditions thereby allowing the causal link to again be assumed. Note that although it is possible to interpret the causal link rule (3.4) as a rule about persistence (a "commonsense law of inertia"), this is not quite accurate. The rules are actually about what constitutes an acceptable plan in the SNLP model. That is, $CLINK(s, P, w)$ states that it is *possible* to use s to achieve P for w. The safety condition rules make this clear, since one would hardly want to form an argument for persistence of P from s to w from the fact that a threat is either promoted or demoted.

Any argument constructed from this defeasible knowledge base of causal link rules is an SNLP plan. It remains to ensure that the plan is complete in the SNLP sense and to deal with the ordering constraints. These constraints represent assumptions that the planner can make, so we add the following assumption to Δ for any steps s and w:

$$\rightarrow s < w.$$

We also have the definition of the initial state in KB:

$$Occurs(START) \supset P_1(out(START)), \ldots, P_n(out(START)),$$

where the P_i are the initial conditions, and the definition of the final state in Δ:

$$G_1(in(FINISH)), \ldots, G_k(in(FINISH)) \rightarrow Occurs(FINISH),$$

where the G_i are the final conditions (goals). Finally, we need to impose the ordering constraints on the initial and final states, in KB:

$$START < FINISH$$
$$\forall s \,.\, START < s < FINISH$$

We could be smarter about these rules, but this specification remains faithful to the orginal SNLP defintions.

With these definitions, a plan for the goals is a well-formed, undefeated, consistent argument for $Occurs(FINISH)$ from $\langle KB, \Delta \rangle$. The fact that the argument is well-formed takes care of the first part of Definition 21 (precondition completeness) since the defeasible rule for $Occurs(s)$ lists all its preconditions. That is, the only well-formed arguments for $Occurs(s)$ that can be constructed from a defeasible knowledge base set up as I have described are those that use the defeasible rule that lists all its preconditions. I deliberately did not add the lattice of less specific precondition rules described in Section 3.3.1, since such partial plans are not part of the SNLP model. If we were to allow them, however, as we surely would want to in a more realistic setting, then we would have to impose the constraint that the argument not have any specializations in order to satisfy the first part of Definition 21. An interesting relaxation of this might be to require validity of the plan instead (*i.e.*, no inconsistent specializations rather than no specializations at all), which would allow unthreatened conditions to go unstated. But as it stands, well-formedness satisfies the first part of Definition 21 because of the form of our defeasible causal rules.

The fact that the argument is undefeated looks after the second part of Definition 21 (threats resolved) since any plan that assumed a causal link without making any necessary safety assumptions would be defeated by an argument that used the more specific threat rule. That is, the threat rule for $CLINK(s, P, w)$ is very strong, and allows any step v, to defeat the basic *CLINK* rule. Of course, the safety rules are provided to resolve this whenever necessary, and unthreatened links can use the basic rule alone. An interesting possibility that this raises is that we may be able to recognize steps that are unordered in the plan. These steps are candidates for parallel execution, as discussed in Section 2.1.3 regarding special cases of simultaneous action handled by nonlinear planners. But the point for now is that the definition of defeat is strong enough to provide the global evaluation required by Definition 21, given a defeasible knowledge base that accurately reflects threats (defeaters) and safety conditions (reinstaters).

Finally, the consistency of the argument ensures that the ordering constraints are consistent, corresponding directly to SNLP's use of order consistency in Definition 22. One lesson we can draw from this aspect of the reconstruction is that while consistency-checking is always going to be a fundamentally hard part of defeasible reasoning, we can look for "islands of tractability" in certain reasoning tasks and use specialized reasoners (such as an ordering constraint propagation system in this case) where they are appropriate.

3.4.3 Examples

Before summarizing what I think has been gained from this reconstruction, two brief examples may help to clarify the translation, as well as illustrating how naturally arguments serve as a representation of plans.

The first example is the simplest possible SNLP plan, represented as an argument. In this plan, the goal is a single proposition, G, that is already true in the initial state. The argument corresponding to such a plan is shown in Figure 3.10. Of course, no actions are required, but the argument does record the important assumption that G is protected from the initial state to the goal state. Such information is what is typically added to plan representations to allow replanning, but is a natural and necessary part of the representation of plans as arguments.

A more complicated example is an SNLP plan for the Sussman anomaly problem from Figure 3.5, an argument for which is shown in Figure 3.11. I have boxed the nodes corresponding to operator executions and abbreviated *START* and *FINISH* to *ST* and *FIN* respectively. The shaded node is an example of the promotion rule being used to protect a causal link.

One thing to note about this argument is its similarity to Figure 3.9, the supported plan for the Sussman problem presented in the previous section. In fact, it would be even more similar except that in that example I glossed the issue of exactly what the goal was and in this one the persistence of the goal conditions until the *FINISH* step is noted. The important diffference, however, is that whereas there are no axioms defining *CLINK* in the knowledge base, the axioms for *NC* impose significant constraints on what events can occur. That is, in the SNLP model, the *CLINK* predicate serves a purely

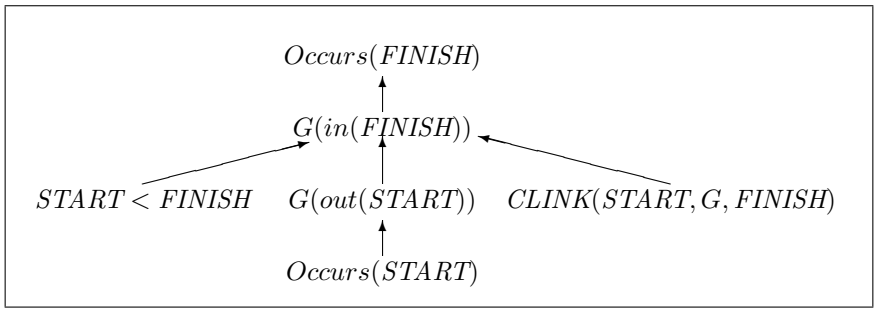


Figure 3.10: The simplest SNLP plan as an argument

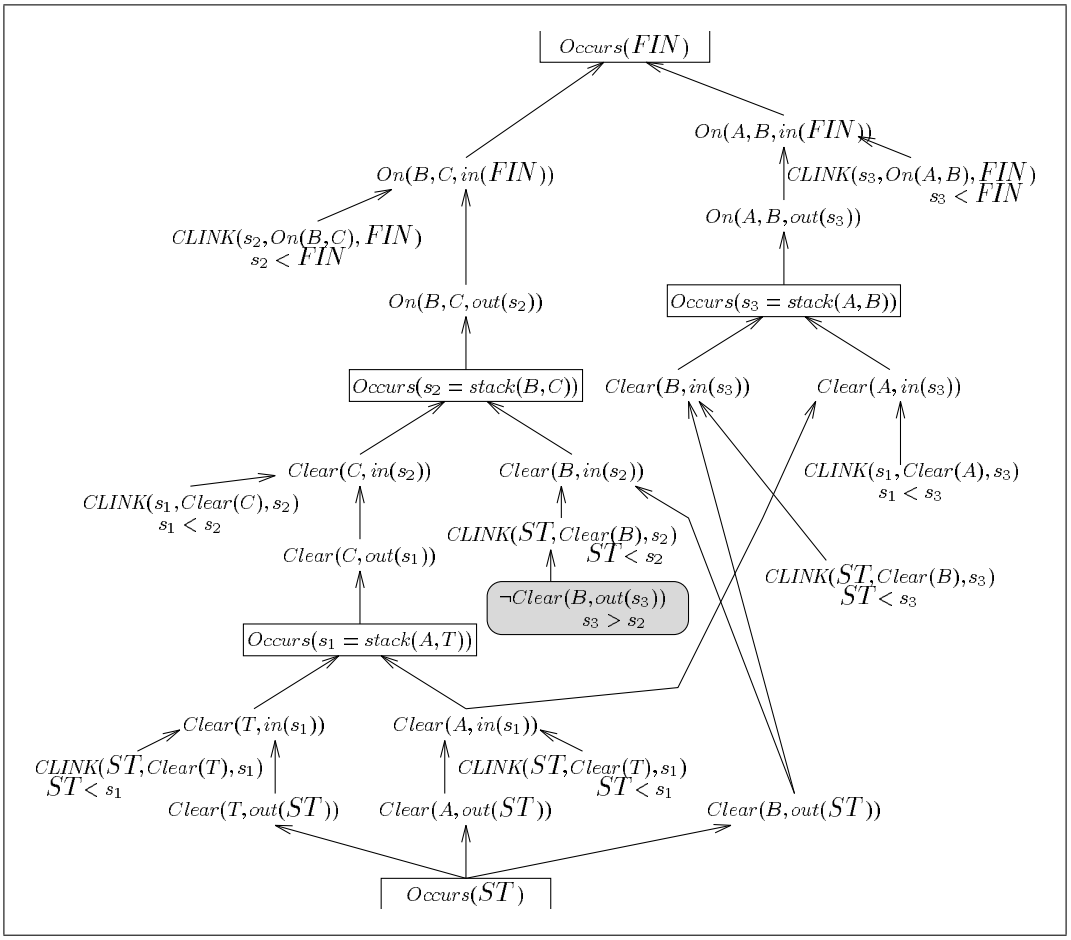


Figure 3.11: Argument corresponding to SNLP plan for the Sussman anomaly

syntactic bookkeeping role, whereas in the more general model based on explanation closure axioms, the NC predicate says something about the world. The difference would arise if we tried to predict other effects of the plan, for example, or answer questions about it, or modify it incrementally. In these situations, which clearly arise in mixed-initiative planning scenarios, the extra epistemic content of the explanation closure-based persistence assumptions is necessary to perform the required reasoning.

3.4.4 Summary

What has this analysis revealed, beyond simply reformulating a simpler reasoner within a more expressive one? I think there are four benefits. First, the defeasible reasoning perspective separates the model of plans and planning from the representation of action. We have used the same defeasible reasoning framework to represent plans based on both the STRIPS model of action and the event-based temporal logic. The separation makes comparison between the resulting planning systems possible.

Second, the reconstruction emphasizes that planning is fundamentally a process of making consistent assumptions, in this case ordering constraints and in the general case assumptions about event non-occurrence. Contrary to the opinion of an anonymous reviewer of a previous version of this work, deciding how to accomplish goals is not the main issue in planning. At least, it's not the hard question—we can get an answer to that by looking at the operator descriptions. What is difficult, as Chapman's famous analysis of conjunctive goals proved although it was intuitively obvious prior to that, is dealing with the interactions between those actions. This is necessarily an assumption-making process, whether it's the STRIPS assumptions about persistence or explanation closure assumptions about event non-occurrence.

Third, the reconstruction illustrates that SNLP's so-called causal links are in fact defeasible connections between steps of the plan. They are only truly "causal" subject to other constraints on the plan. By making the defeasible rules and assumptions underlying the definitions explicit as part of a more general reasoner, we can see more clearly exactly what they are and what role they play in the formation and evaluation of plans. The fact that the threat rule and the promotion and demotion safety rules seem somewhat awkward is some evidence that SNLP's conception of a causal link may not be entirely intuitive. Note also that evaluation of the plan is based on a global evaluation of both the structure of the plan (the rules) and the assumptions on which it is based (the ordering constraints). Again, the argument-based formalism is ideally suited to such reasoning.

Finally, the reconstruction applies argument-based reasoning to an important problem, namely STRIPS planning, which despite all its shortcomings remains a subject of active research in AI. More than just respect is gained, however. The significant body of work on algorithms for STRIPS planning, primarily heuristic search techniques, can be imported and used in the construction of defeasible reasoners. As well, the work on reactive or "anytime" planning applies here in the context of resource-bounded, "ampliative" defeasible reasoning. Much of this remains for future work on mixed-initiative planning systems, although Chapter 4 describes our implemented plan reasoning system

based on arguments but using heuristic search. I think a good place to start furthering the connection between defeasible reasoning and planning would be a reconstruction of Sacerdoti's "critics" (Sacerdoti, 1977) (or their derivatives) in terms of argument preference criteria. I think the reconstruction would both illuminate the prior work and yield benefits for future planners.

3.5 Discussion and Related Work

This section considers related work on representing plans and reasoning about them from several disciplines, both within AI and outside it. In some cases the representation of plans-as-arguments has clear advantages. In others, the difference is not so much one of content as of perspective, in which case the work is often complementary to that presented in this chapter. I will partition the discussion into four categories: nonmonotonic reasoning, traditional planning, planning and discourse, and computational dialectics.

3.5.1 Nonmonotonic Reasoning

I want to start where the previous chapter left off, in a sense, namely with work on nonmonotonic reasoning about action and change. As that chapter made clear, there is a significant amount of work on using nonmonotonic formalisms for prediction and, sometimes, explanation. Aside from the problems discussed there, the question I want to address here is whether this work can be applied to the third reasoning task listed at the outset of Chapter 2, namely planning, or more generally, reasoning about plans. The answer, surprisingly, is that it really has not been. I think the reason for this is that, as I have stated repeatedly, the emphasis in "classical" nonmonotonic reasoning is on the conclusions that should be drawn from a defeasible knowledge base absent any other constraints. I have argued that planning and plan reasoning does not really fit that form of reasoning, except in the trivial sense of generating possible complete plans and asking the nonmonotonic reasoner if they are "correct," whatever that means (*i.e.*, plan validation, see (Nebel and Bäckström, 1994) for an interesting discussion of the relationship between planning and validation). I emphasized in Section 3.2.4 that argumentation is concerned with the process of drawing conclusions, rather than just what those conclusions should be. Similarly, planning (and plan reasoning in general) is concerned with forming plans, not just in verifying that a particular plan will succeed.

There is some work in the nonmonotonic reasoning community on planning that I think makes this clear. For example, Elkan (1990) presents an algorithm for incrementally constructing plans based on a nonmonotonic logic. The scope of the proposal is limited by the use of a Horn clause logic and the fact that the only assumptions permitted are negative preconditions (*i.e.*, defeaters can be assumed not to defeat). These restrictions allow a straightforward implementation in Prolog, since negation as failure ensures that proving a negative subgoal can never add information to a proof that previously assumed it. These limitations make it difficult to see how the approach might be extended to a representation of action such as the one presented in Chapter 2. The

emphasis on incrementality, however, is one of the features of argument systems and not of nonmonotonic logics by themselves.

I noted previously (Section 3.2.4) the intellectual closeness of the argument-based approach to defeasible reasoning and the approach based on theory formation (Poole et al., 1987; Poole, 1988). The application of theory formation to planning in particular is presented in (Goebel and Goodwin, 1987). Although much of the discussion of action representation and reasoning about change parallels that presented here, planning is treated as a process of competing predictions (for the goal and its negation) being resolved using theory preference knowledge. This is similar in spirit to my approach based on explicitly constructing arguments.

Finally regarding traditional nonmonotonic reasoning approaches to plan reasoning, one of the most successful such efforts is Kautz's formalization of plan recognition in terms of circumscription (Kautz, 1987, 1991). One reason for this is that it made concrete an aspect of plan reasoning that had formerly only been studied informally in terms of linguistic theory and pragmatics. The cost of such formalization, however, was a restriction of the scope of the recognition to very simple ontologies. For example, although there were decomposition and abstraction hierarchies in the representation of events, there was no ability to reason about propositional effects of actions and how to achieve goals. And again, the use of circumscription meant that the resulting system was more a specification of what acceptable plan recognition was than it was a way of computing them, although the well-described implementation (Kautz, 1991) probably deserves more credit than this. As with circumscription in general, the fact that the important information contained in the circumscription policy is outside the language limits its usefulness for mixed-initiative planning where we need to be able to discuss these issues. An interesting application of Kautz's approach in an interactive planning system that does consider these issues is presented in (van Beek et al., 1993).

3.5.2 Planning

Moving from nonmonotonic reasoning in the abstract, the next related area is the large body of work in AI planning. We emphasized in Chapter 2 that for the most part these systems are based on a temporal representation and ontology of actions that is inadequately expressive for realistic mixed-initiative planning. The conclusion from this chapter, however, is that even if that aspect were somehow to be fixed, the requirement that plans be deductively guaranteed to achieve their goals is not only unreasonable but undesirable. The reconstruction of SNLP in Section 3.4 illustrated how to fit traditional planning into the defeasible reasoning framework. The conditions that had to be imposed on the resulting plans-as-arguments were very strong, however, corresponding to the deductive guarantees of STRIPS plans. We noted several places where relaxing these constraints might lead to interesting types of plans.

Work on traditional planning becomes more interesting when it is extended beyond the original toy domains and black-box planning problems. For example, the work on plan modification and replanning (Kambhampati, 1989; Kambhampati and Hendler, 1992) is based on a representation of plans (PRIAR) that annotates them with causal

dependency links. This additional information turns out to be identical to the information contained in arguments about plans. For example, the Blocks World stacking example in (Kambhampati, 1994) has validation links corresponding to the *CLINK* rules of the SNLP plan in Figure 3.11 or to the *NC* assumptions of event non-occurrence in Figure 3.9. The common observation is that it is necessary to make the assumptions underlying the plan explicit if we are to be able to reason about interactions between them. The validation links of the PRIAR model are one class of assumptions and, as shown in the previous section, are sufficient for STRIPS planning. While the general defeasible reasoning approach is more general, we can hope to use the algorithms developed for replanning when they apply.

One of the important aspects of the representation of plans based on arguments is that they allow partial plans to be described and evaluated. In the formalization of the STRIPS model due to Chapman (1987), partial plans are given meaning only in terms of their set of completions. We saw in Section 3.3.3 that this is only one dimension along which to characterize partial plans (and corresponds to my notion of validity). As well, the fact that partial plans can arise naturally in mixed-initiative planning and be reasoned about without reference to their completions is an important aspect missed by the TWEAK model. There has been work on partial STRIPS plans in the context of abstraction (Yang and Tenenberg, 1990; Knoblock, 1992), but it is not clear that abstraction as developed in ABSTRIPS is the same thing as the kind of incompleteness found in argumentation.

Finally, Ginsberg (1994a; 1994b) has recently argued that planners, instead of being guaranteed correct and complete, should be approximate, a term which he defines formally. The representation of plans as partially-ordered sequences of actions is significantly less expressive than we have claimed is needed for mixed-initiative planning. The interesting aspect of this work for my purpose, however, is that for the main result regarding convergence of a planner to the optimal plan, Ginsberg uses a dialectical construction identical to that used in argumentation. That is, starting from the empty set of plans, the planner first constructs a set of plans that achieve the goal(s) that have not yet been considered, then considers exceptions to those plans. This process repeats and, under certain fairly strong conditions on the sets of plans produced at each step, will converge to the set $L(g)$ of all linear plans that achieve the goal g. Also as in my presentation, the exact structure of plans is not dependent on a particular action representation such as STRIPS (although it does characterize plans as simply sequences of actions, so it's not clear whether it is sufficiently expressive). It remains to be seen whether this development will be pursued by the planning community.

3.5.3 Planning and Discourse

Several researchers have been motivated by the role of plan reasoning in discourse, and much of this is based on the same motivations as I have presented for mixed-initiative planning.

The connection between planning and discourse was initially noted in work by Cohen (1978; 1979) and Allen (1979; 1980; 1983b) in applying AI planning techniques to the

theory of speech acts developed by Austin (1962) and Searle (1969). They noted that if speech acts (*i.e.*, things that can be done by speaking) are represented as planning operators with preconditions, effects, and so on, then it is possible to plan sequences of utterances to accomplish discourse-level goals, such as the speaker informing the hearer of something. Similarly, the hearer could use the same knowledge to determine the intentions of the speaker from her speech acts by recognizing her plan (Litman, 1985; Litman and Allen, 1987). The most intriguing part of Allen's work as regards my development of plans-as-arguments is the "rating heuristics" applied to plans during plan inferencing. Since the representation of plans is based on an augmented STRIPS model, these heuristics can only be specified informally and implemented heuristically. One of the reasons for moving to a representation based on arguments and argumentation is that it allows such properties of plans to be expressed formally and reasoned about as part of the reasoning about plans.

Starting from this work on plan recognition for discourse processing, Pollack (1986a; 1986b; 1990) noted that in order to properly account for certain phenomena, for example recognizing incorrect plans, a more sophisticated model of belief and intention was necessary. The result was a view of plans as complex mental attitudes. In (Konolige and Pollack, 1989) this is combined with Konolige's (1988) system of explicit defeasible reasoning discussed in Section 3.2.4. The motivation is that plan recognition is a process of belief and intention ascription that is necessarily defeasible. Konolige and Pollack argue that there will often be conflicts between possible ascriptions and that the interesting part of plan recognition comes in the adjudication between these conflicts. Argumentation systems such as theirs or the one described in this chapter are ideally suited to representing such knowledge and performing such reasoning in a principled manner. I see the difference between this work and my own as one of focus rather than content. Building on the representation developed in Chapter 2, I have been concerned with domain-level plans, *i.e.*, arguments about courses of action and their relation to the world, in a language that is sufficiently expressive to represent complicated, realistic domains. The work on plans as complex mental attitudes concentrates on the beliefs and intentions of agents about such actions. That is, whereas they reason defeasibly about plans, for me the plan is itself a defeasible reasoning structure. These viewpoints are certainly not incompatible and, given the other similarities, can probably be combined in a straightforward manner.

I would like to also note here Pollack's excellent IJCAI Computers and Thought Award lecture on the uses of plans (Pollack, 1992). Many of the motivations I have given for my work are well laid out there, including the critical observation that, the history of AI notwithstanding, "agents use plans not only to guide action, but also to control reasoning and enable inter-agent coordination" (page 44). Pollack also notes the philosophical, empirical, and engineering problems that result from attempting to design intelligent systems that do not use plans.

The use of plans in collaborative behavior is the basis of the SharedPlan model (Grosz and Sidner, 1990) that extends the notion of plans as structured sets of beliefs and intentions to situations where several agents are collaborating. In its original formulation, the domain plan (termed a "recipe" in (Pollack, 1986a)) was restricted to

very simple sets of events related via event generation. It is clear that this is insufficient in general, especially when the subject under discussion is a plan, as it is in collaborative problem solving and mixed-initiative planning. Balkanski (1990) described a more complex formalism for describing recipes that included event enablement and composition, sequencing, and iteration operators. There is, however, still no way to reason about states and their relation to actions as in traditional planning. And although the language allows more complicated recipes to be constructed, the question of reasoning about these recipes at the domain level is not addressed. It seems to me that many of the issues discussed in Sections 2.4 and 2.5 regarding reasoning about action in complex domains will arise within this framework. The defeasible nature of this reasoning and the role of assumptions during reasoning about recipes (plan reasoning) is also not addressed.

Plan recognition based on the SharedPlan model augmented with the more expressive recipe language is described in (Lochbaum et al., 1990). The goal is to describe how the structure of the recipes influences recognition. The idea is that the recipe can be partially specified during construction by the agents, based on a relation on act-types that they name "Contributes." This is similar to the way that plans-as-arguments can be partial and can be incrementally refined, and the "Contributes" relation corresponds to the way agents evaluate plans-as-arguments relative to their knowledge of causality. In fact, "Contributes" is formalized as the transitive closure of the various one-step links that appear in the act-type relation ontology. Lochbaum *et al.* note that the belief ascription process is defeasible, and indicate that argumentation as in (Konolige and Pollack, 1989) is one way of performing such reasoning. Lochbaum (Lochbaum, 1991) describes an algorithm for explaining the role of a particular activity in an augmented recipe. The approach seems very similar to the constraint-based search used by my implementation of recognition (*cf.* Section 4.5.2). Again, I think that the use of the formal system of defeasible reasoning allows a more principled specification of the kinds of reasoning being performed. However, as was the case with plans as structured intentions, there are many aspects of the work on the SharedPlan model that need to be incorporated into my approach.

Finally, some of the work most closely related to mine on mixed-initiative planning is Carberry's work on collaborative information-seeking dialogues (Carberry, 1990). This is based on a "context model" consisting of a tree of goals with associated plans, and Carberry shows how it can be used to handle some complicated discourse phenomena such as ill-formed queries and inter-sentential ellipsis. In (Lambert and Carberry, 1991) this is extended to a three layer model involving plans at the domain level, the problem-solving level, and the discourse level. Ramshaw (1991) proposes a similar three-layer model. My work on plans-as-arguments has so far concentrated on only the domain level, although the relationship between plans and discourse actions has been explored in the context of the TRAINS project described in the next chapter (see also (Traum and Allen, 1994; Traum, 1994; Ferguson and Allen, 1993)). As I noted previously, arguments provide a framework for recipes (plans) that is suitable for both the linguistically complex discourse phenomena and the complicated domain-level reasoning. I think this latter aspect is not considered in as much detail in the work on information-seeking

dialogues. Furthermore, by formulating plan reasoning as argumentation, it may be possible to formalize the problem-solving plans of these tripartite models, rather than simply accepting the sample plans used in examples.

Regarding defeasibility, Carberry (1991) presents an approach based on using Dempster-Shafer theory to represent the support that evidence about the user's actions gives to alternative conclusions about her goals. Default reasoning is then performed by preferring choices with the highest "plausibility factor" when ambiguities arise. This direction seems very promising, although in its current formulation it requires large numbers of prior probabilities (*cf.* Section 3.2.4). It also may not admit the kind of qualitative default reasoning that arises in reasoning about plans (for example, when parts of plans are not yet specified). As well, I believe that that the argumentation approach is appropriate for expressing the kind of meta-knowledge and heuristics that Carberry indicates are necessary for resolving problems between the participants.

Finally on this topic, I would like to note the similarities between the model of collaboration presented in (Chu-Carroll and Carberry, 1994) and my approach based on argumentation. Their "Propose-Evaluate-Modify" cycle is exactly the dialectical process underlying argument-based reasoning. They note that it is recursive, since the modification phase can involve collaboration. Similarly, argumentation about a domain phenomena can lead to argumentation about the rules supporting the conclusions, and thence to the preferences on rules and principles for argumentation. The "evaluator" component of their system seems very much like the properties of plans-as-arguments described in Section 3.3.3, even using terms such as "well-formedness" and "infeasible" (*i.e.*, unsupported) actions.

In conclusion, as I noted at the outset of this section, much of the work on planning and discourse is motivated by the same interest in intelligent, collaborative reasoning that motivates my work on mixed-initiative planning. That other work has generally focused on discourse-level phenomena, particularly in the context of recognition. I have concentrated on developing a model of plans at the domain level that combines work on AI planning and reasoning about action with the sort of direct, defeasible reasoning required for mixed-initiative planning. I do not see the two areas as competitors. I think that the representation of plans-as-arguments has proved useful both at the domain level for reasoning about the effects of actions and at the discourse level for reasoning about how utterances contribute to a dialogue. I think it may also allow a clearer formalization of some of the procedural and heuristic techniques underlying the work in planning and discourse. On the other hand, the reasoning about beliefs and intentions and the connections between discourse phenomena and domain plans is something that I hope can be effectively imported in future work on representing plans using arguments.

3.5.4 Computational Dialectics

The study of defeasible reasoning has a rich intellectual tradition dating back to long before nonmonotonic reasoning became fashionable in AI. As a technical form of reasoning, argumentation dates to Socrates and the ancient Greeks, although presumably people started arguing (informally) almost as soon as they started talking, if

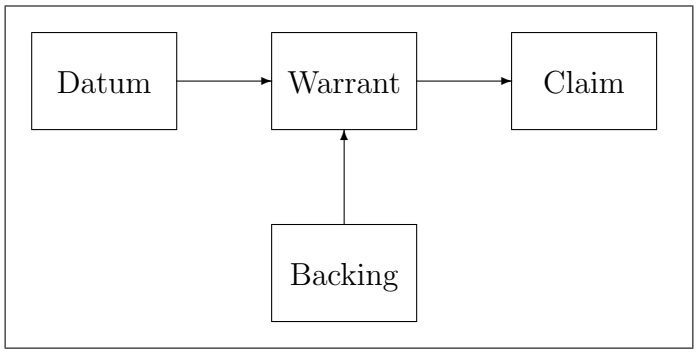


Figure 3.12: A Toulmin diagram

not sooner. Loui (1991a), in his excellent survey of the history of non-demonstrative argument-based reasoning, credits Keynes with reviving its formal study, citing such passages as "probability is concerned with arguments" (Keynes, 1921, page 116).

The field of computational dialectics is an emerging union of work on nonmonotonic reasoning, AI and law, computer-aided instruction, philosophy of language, and philosophical logic. To paraphrase Tom Gordon's description at the AAAI-94 Workshop on this topic, its subject is the design and implementation of computer systems which mediate group decision-making processes under conditions that include resource limitations and divergent knowledge, interests, and values, whose goals include such normative criteria as rationality, effectiveness, and fairness, and whose agents can be human, artificial, or both. In this brief survey I will only touch on a small part of the work in this field and its intellectual precursors that I have found particularly relevant to the application of argument-based reasoning to reasoning about plans.

Proceeding chronologically, Toulmin (1958) presented a model of argumentation that is more structured than the rule-based model I described in Section 3.2. Figure 3.12 illustrates the four elements that, according to Toulmin, make up an argument. The warrant corresponds to the rule used to reason from the datum to the claim, and the backing is the authority or justification for the rule itself. The main feature of Toulmin diagrams is that they can be chained at any of the three terminal nodes, corresponding to chaining (at the datum and claim nodes) and to recursive argumentation (at the backing node). Regarding plan representation, these diagrams are interesting as a perhaps more intuitive representation of an argument. Accordingly, they are being used in the design of user interfaces for computer-aided instruction systems (*e.g.*, (Cavalli-Sforza and Suthers, 1994)), although these applications generally do not do much reasoning about what they are displaying. As the development of mixed-initiative planning systems starts to include additional modalities of communication such as graphical user interfaces, however, there will need to be more interaction between these communities.

One of the important aspects of computational dialectics that I have not considered in detail is the role played by a specific communication protocol. This is most clear in system of argumentation developed by Rescher (1976; 1977) which builds on previous work in formalizing non-demonstrative reasoning as a "game" in which players can make

various "moves." Rescher also introduced the notion of a formal defeasible rule, even going so far as to use the now-canonical "birds fly" example to illustrate it. This type of system has been particularly popular in work on formalizing legal reasoning. Gordon (1993) presents a comprehensive model of adversarial legal argumentation during the pleadings phase of a trial. In addition, he presents a detailed exposition of related work on nonmonotonic and argument-based reasoning, in particular emphasizing the process-oriented view of reasoning afforded by the dialectical perspective. I have also made this point repeatedly.

The key difference between systems such as these and the kind of mixed-initiative planning systems I am envisioning is that it is not always clear in language-based mixed-initiative systems exactly what "move" is being made by a participant. In Gordon's Pleadings Game, the moves are terms of a language including operators such as `claim`, `concede`, `deny`, and so forth. But in systems such as I described regarding planning and discourse, the contribution of a particular element of the interaction, for example an utterance or an observed action, needs to be determined by the observer. Typically this determination is ambiguous, leading to the defeasibility of the reasoning process in addition to the fact that what is being reasoned about is the application of defeasible rules. One approach that I will touch on again in Section 3.6.1 is that, at least initially, we might consider restricting the set of "speech acts" from the very linguistically-oriented ones used in the planning and discourse work towards some more literal, less ambiguous ones. This is easily accomplished through the use of a graphical user interface, perhaps in combination with a simplified natural language query language. This would perhaps allow us to use the work on protocols for language games more directly, or adapt the informal protocols such as Gricean maxims (Grice, 1975) to the formal argument system.

Gordon makes another point regarding the relationship between argumentation and nonmonotonic reasoning that I think is worth paraphrasing. He notes that it has become popular to justify a new form of nonmonotonic reasoning by defining a semantic model theory and showing that certain desired conclusions are indeed entailed using it. But, he argues, the model theory itself must then be justified using "philosophical arguments and examples which, together, persuasively support the claim that it has captured the intuitive meaning of plausibility." (page 97) And such arguments can also be offered in defense of the argument-based "proof theory" directly, so it is not always the case that the model theory has helped matters. In some cases it may, but it is not necessary for an appreciation of the use of the formalism for reasoning.

Finally, within the philosophical logic community there has been interest in various kinds of informal logic, including argumentation. Walton (1992) provides a good survey of the field in addition to developing an almost computational model of such reasoning. For example, he considers the issue of resource bounds and the implication of them for satisficing versus maximizing reasoning, a topic familiar to AI through the work of Simon (1977) and almost any aspect of AI since. He also touches on issues relating to the linguistic study of argumentation and defines twelve types of dialogue including "debate," "inquiry," "planning committee," "quarrel," and "deliberation." Walton also criticizes the "semantic conception of argument" (pp. 167–170) akin to the preceding

paragraph.

There is much more to computational dialectics than what I have noted here. Once we start thinking of the reasoning being performed by an agent as a *process*, either external (involving communication) or internal (involving deliberation) or both, the connections to other fields appear very quickly. I think it is important that AI, and the planning community in particular, not ignore these perspectives as we work towards theories and systems for mixed-initiative planning.

3.6 Future Work

As I have emphasized throughout this chapter, the work on mixed-initiative planning in general and on the application of the representation of plans as arguments is really only just beginning. The next chapter describes my other work on mixed-initiative planning in the context of the TRAINS system, so in this section I would like to present three outstanding issues regarding the argument-based plan representation. I hope that the fact that they all seem at least plausible given the development so far lends support to the thesis that plans should be thought of as and represented by arguments. The three topics are: a plan description language, graph-based utility models, and general evaluation and preference criteria.

3.6.1 A Plan Description Language

The first issue is one that was originally a major focus of my work but that I was forced to leave only partially studied. This is the idea that we use arguments as the denotations of terms that represent plans in a formal plan description language. The motivation for this is that, as observed in Section 3.1.2 regarding mixed-initiative planning, we need to be able to talk about and refer to plans and parts of plans and their relationships. For example, when the manager says something like "How does that (plan) sound?" the translation to our representation language needs to have a term denoting "that plan," which would need to be resolved just like "that engine" or "that boxcar." Or in a graphical user interface, the manager could select part of a plan (presumably from a display that looked very much like the arguments we've been drawing, but not necessarily), select a type of event with certain parameter settings, and indicate that the event should be added in service of the specified goal. Again, a principled translation of these actions into our interface language requires a language that refers explicitly to plans.

In case that sounds like too "linguistic" a reason to want a representation language with plan-denoting terms, consider that such a language could also serve in the formal specification of plan reasoning algorithms. We could then verify the algorithms against the formal specification, or perhaps even "execute" the specification directly in the same way that Chapman's modal truth criterion corresponds to a nondeterministic nonlinear planning algorithm. A similar use would be in the expression of evaluation and preference criteria as will be described in Section 3.6.3. In all three cases, the

representation of plans based on arguments provides exactly the conceptual repetoire we need to express properties of plans in such a language.

Unfortunately, I have not explored this beyond some fairly trivial first steps. It is clear that given a logical language with quotation, we can represent plans-as-arguments using structural statements such as:

plan-event(e, p)	*plan-enables*(e, ϕ, p)
plan-fact(ϕ, p)	*plan-achieves*(ϕ, e, p)
plan-premise(ϕ, p)	*plan-generates*(e_1, e_2, p)
plan-goal(ϕ, p)	*plan-supports*(ϕ_1, ϕ_2, p)

In these formulas, the e_i denote events, the ϕ_i are quoted sentences, and p denotes a plan. We can then give these statements the obvious semantics by defining an interpretation $\mathcal{I}$ that maps plan terms to arguments and using the definitions and properties of arguments from Sections 3.2 and 3.3. For example, if we partition the set of propositions used in an argument A, $Props(A)$, into two sets, $Events(A)$ and $Facts(A)$ in the obvious manner, we can then define the enablement relation:

$$\begin{aligned}
&\models_{\mathcal{I}} \textit{plan-enables}(\phi, e, p) \text{ iff} \\
&\quad \mathcal{I}(\phi) \in Facts(\mathcal{I}(p)) \text{ and } \mathcal{I}(e) \in Events(\mathcal{I}(p)) \text{ and} \\
&\quad \exists \Phi \,.\, \langle \Phi, Occurs(e) \rangle \in \mathcal{I}(p) \wedge \phi \in \Phi.
\end{aligned}$$

This is only so-called "direct" enablement, but we could equally well capture enablement through multiple steps.

There are many technical issues here, and I don't want to promote this particular definition too much. I just want to emphasize that it is the sort of thing one can say. The claim is that the sorts of things we want to say about plans are easily translated into a language based on arguments. We could base it on uninterpreted graphs, but I think we would then simply reinvent much of the vocabulary of arguments in order to be able to express the desired properties of plans. In any case, we can import whatever graph-theoretic notions we need into the argumentation framework.

There are two directions in which to pursue this. First, we can try to formalize the definitions of defeat, *etc.*, axiomatically using sentences of the meta-language, its "is true" predicate, and some set theory. In previous work (Ferguson, 1992), I did this for a similar representation of plans that was not based directly on arguments but on less well-founded "plan graphs." The exercise really didn't illuminate what was interesting about the representation, although it did result in an "executable specification" (which would never halt in practice, of course).

So then the other direction is to return to the "linguistic" motivations and develop the plan description language based on the needs of a plan reasoner's interface. The next chapter presents some informal but implemented steps in that direction in the context of the TRAINS domain plan reasoner. The next generation of TRAINS will be based on a set of generalized speech acts that is both more liberal, for instance including mouse gestures and menu selections, and more literal in terms of content. The hope is

that with a more constrained and less ambiguous communication protocol, we will be able to formalize more effectively the actions that make up the mixed-initiative planning interaction. These actions will be based on the representation of plans as arguments and will determine the structure of the plan description language. Additionally, more precise specification of the way plans are output (displayed as graphs, described in language, simulated on a map, *etc.*) will similarly impose additional constraints on the plan description language.

3.6.2 Graph-based Utility Models

The second major area of future work in which I see the plans-as-arguments formalism being successful is an integration with existing work on graph-based probability or utility models. I noted at the end of Chapter 2 that such a capability was obviously needed to reason in realistic domains. Defeasible reasoning already allows some forms of uncertain reasoning to be captured more naturally and effectively than the deductive logic presented there. As I stated in Section 3.2.4 when comparing defeasible reasoning and decision theory, one way of viewing defeasible rules is as qualitative conditional probability statements. Specificity and defeat can then be justified by appeal to principles of statistical inference such as (Kyburg, 1974). But I think it's clear that we need more quantitative statistical reasoning in order to reason effectively about plans.

One of the most promising recent developments in uncertain inference has been the study of Bayesian networks (Pearl, 1988). Recall that the nodes of a Bayesian network represent variables under analysis and the links represent conditional dependence. Importantly, the absence of a link between two nodes implies their conditional independence. Belief in propositions can then be effectively computed given appropriate prior distributions and observations. In the context of causal reasoning such as we are concerned with in planning, the nodes of the network represent events and propositions and the links encode the "flow of causality," again seen as conditional dependence. In the symbolic causal networks of (Darwiche and Pearl, 1994a,b), this "causal structure" is augmented with a set of "domain micro-theories" that specify logical relationships between propositions and their direct causes. The results look very much like arguments. The causal network is a directed, acyclic graph labelled by propositions and whose links denote a form of influence. The domain theory expresses non-defeasible knowledge about the domain. In fact, Darwiche and Pearl (1994a) note that symbolic causal networks compute "arguments" in the ATMS sense.

The shortcoming of any of the Bayesian network-based reasoning models is that they generally presuppose the existence of the network. But if we want to represent plans using these structures, then the issues we have raised previously regarding mixed-initiative planning require that some account be given of how these networks might be formed, recognized, and so on. There has been work on learning networks (*i.e.*, causal models) from observations (Sprites et al., 1990; Jearl and Verma, 1991; Lam and Bacchus, 1994) but this is generally based on large numbers of observations of regularly-occurring phenomena rather than construction of something like a plan in a particular situation, with particular partners, and so on. The connection I see between

argumentation and Bayesian networks is that the theory of defeasible reasoning describes the dialectical process that goes into constructing a network (call it a model, or a plan, or an argument). That is, we argue (perhaps with ourselves) about what is salient and what the dependencies are. Then one of the ways of evaluating the argument is to evaluate it as a Bayes net or symbolic causal network. The values of certain nodes, such as the goals of the plan, would be part of the evaluation of the plan as a whole. But this would only be one dimension for evaluation, and the idea of arguments interfering with and being preferred over one and other allows other metrics to be applied as well. I think this is more realistic than the treatment of plans in (Darwiche and Pearl, 1994b) that basically replicates the network for each time step and allows cross-links to represent temporally-sequenced causality. Although this may work for Yale shooting problems and using the situation calculus, it will be expensive, unnatural, and not extensible to richer temporal ontologies.

3.6.3 Evaluation and Preference Criteria

The discussion of Bayesian network evaluation of arguments leads to the final topic I want to mention, namely the specification of evaluation and preference criteria. In the definition of conflict between and defeat among arguments (Section 3.2.2), we defined the specificity criterion for defeat and noted that others were possible. Perhaps Kyburg (1974) is right and what we need are his three fundamental principles of statistical reasoning (specificity, Bayesian construction, and supersample) (Kyburg Jr., 1991). Or perhaps the appropriate set of principles is that described by Pollock (1991; 1992a; 1992b), including some classical logic, the statistical syllogism, some induction principles, and an analysis of phenomena such as collective defeat and self-defeat. I should also mention the work of Goodwin (1991) that adds a language for specifying theory preference and pruning knowledge to the theory formation approach to defeasible reasoning.

In any case, many of the evaluation criteria we need for reasoning about plans are wholly domain-dependent. We might prefer a plan that uses the least time, or transports the most cargo, or has the highest expected utility, or minimizes the chance of some particularly awful event occurring, and so on. And not only do we need to be able to write down such preferences, we also need to be able to communicate them, recognize them in others both when they are overtly stated and when they are implicit in their other actions, and, importantly, argue about them. This requires a language for specifying preferences over arguments and a communication protocol for arguing about them. Although such an ability is obviously necessary for building the kind of interactive, intelligent planning systems we ultimately want, I have not even started to work on it. The work on formal dialectical models mentioned previously is probably the best place to start. For example, Gordon's model of legal reasoning (Gordon, 1993) allows discussion of rules, their backing, the meta-rules governing such discussion, and so on. Something similar is needed for talking about plans, the defeasible causal rules, the persistence assumptions, properties of plans, planning strategies, and so on.

3.7 Summary and Contributions

This chapter has made two main contributions. First, I have presented a representative example of a mixed-initiative planning scenario gathered experimentally and used it to draw several conclusions about aspects of mixed-initiative planning that we need to take into account if we are to build systems capable of participating in such interactions. These conclusions are, first, the communicative nature of mixed-initiative planning, with the corollary that unless we take communication seriously in our representations of plans we will not be able to build natural interactive systems. Second, I noted that the reasoning that goes on in mixed-initiative planning is fundamentally defeasible, both because the domains are more realistic and hence less certain, but also because communication imposes uncertainty, as does the fact that deliberation is resource-bounded in interactive systems.

To accomodate these conclusions, I have described an approach to representing plans by treating them as arguments in a system of defeasible reasoning that explicitly constructs arguments from defeasible rules. I have shown how to address the qualification problem using defeasible causal rules that weaken the meaning of preconditions from necessary and sufficient conditions to causal "influences." I have shown how to address the frame problem by adapting the explanation closure technique of the previous chapter to provide the basis for assumption-based persistence reasoning. These techniques were illustrated with both a series of formal definitions relating abstract properties of arguments to intuitive properties of plans, and with a reconstruction of traditional STRIPS planning within the defeasible reasoning paradigm. The former provides a basis for further formalization and the latter, I believe, shows the naturalness of the approach as well as the fact that it subsumes the representational aspects of traditional planning work. Although many questions are left unanswered, I hope the discussion of related and future work has helped increase the plausibility of the approach.

In the final analysis, I think that what is most important is that the representation of plans and the reasoning about them is based on a form of reasoning, argumentation, that is motivated by both communication and defeasibility. The dialectical nature of argumentation is ideally suited to both the form of the interaction between agents in a mixed-initiative situation and to the reasoning of a single agent in a resource-bounded interactive situation. Arguments make explicit the very concepts we are trying to formalize, such as rules, support, conflict, and defeat, which again gives them an advantage over indirect methods for representing plans. The investigation of mixed-initiative planning and formal models to represent it has only just begun, so I cannot make any great claims. I do think that it is more important to set one's sights high and aim for a natural, comprehensive representation, as I have done, than it is to artifically restrict the data under consideration in order to prove theorems. Or, to quote Martha Pollack, "theories before theorems." (Pollack, 1992, page 65) The next step is to build systems based on the theories and take what lessons we can from them in refining the theories, which is the subject of the next chapter.

4 The TRAINS-93 Domain Plan Reasoner

This chapter describes the TRAINS-93 domain plan reasoner, which is based on the logic of time, events and actions presented in Chapter 2 and on the representation of plans as arguments presented in Chapter 3. This chapter describes the implementation in detail, serving as both a case study in building a sophisticated plan reasoning system to support mixed-initiative planning, and as a guide for further development.

The chapter begins with an overview of the TRAINS system in order to place the domain plan reasoner in context, and then outlines the functionality that the domain plan reasoner module must provide as part of the overall system. The implementation of the event-based temporal logic as a Lisp "toolkit" is presented next, followed by the representation of plans and the implementation of the plan reasoning algorithms. The TRAINS-93 sample dialogue is then presented and analyzed in detail to illustrate the operation of the entire system, concentrating on the plan reasoner and its interaction with other modules. I conclude with a discussion of design issues and future work, followed by a summary of the system and its contributions. Appendices provide detailed descriptions of the functions and data structures used in the implementation.

4.1 TRAINS System Overview

The TRAINS project is on ongoing effort to build an intelligent planning assistant that is conversationally proficient in natural language. The overall system architecture is shown in Figure 4.1. This section describes the role of each module and the flow of information between them. A more detailed description of the system, especially the language-oriented components is presented in (Allen et al., 1995). Recall from the example in Section 3.1.1 and Figure 3.1 (page 63) that the TRAINS domain is a transportation world with cities, rail links between them, engines, boxcars, and the like to move things around, and a variety of commodities.

Natural language input is first processed by a GPSG-style parser into a preliminary unscoped logical form. This logical form is expressed in Episodic Logic (Hwang and Schubert, 1993a,b), a highly-expressive situational logic developed expressly for the purpose of representing the content of natural language utterances and reasoning about them. Among the features of episodic logic are: generalized quantifiers, lambda abstraction, sentence and predicate nominalization, sentence and predicate modifiers,

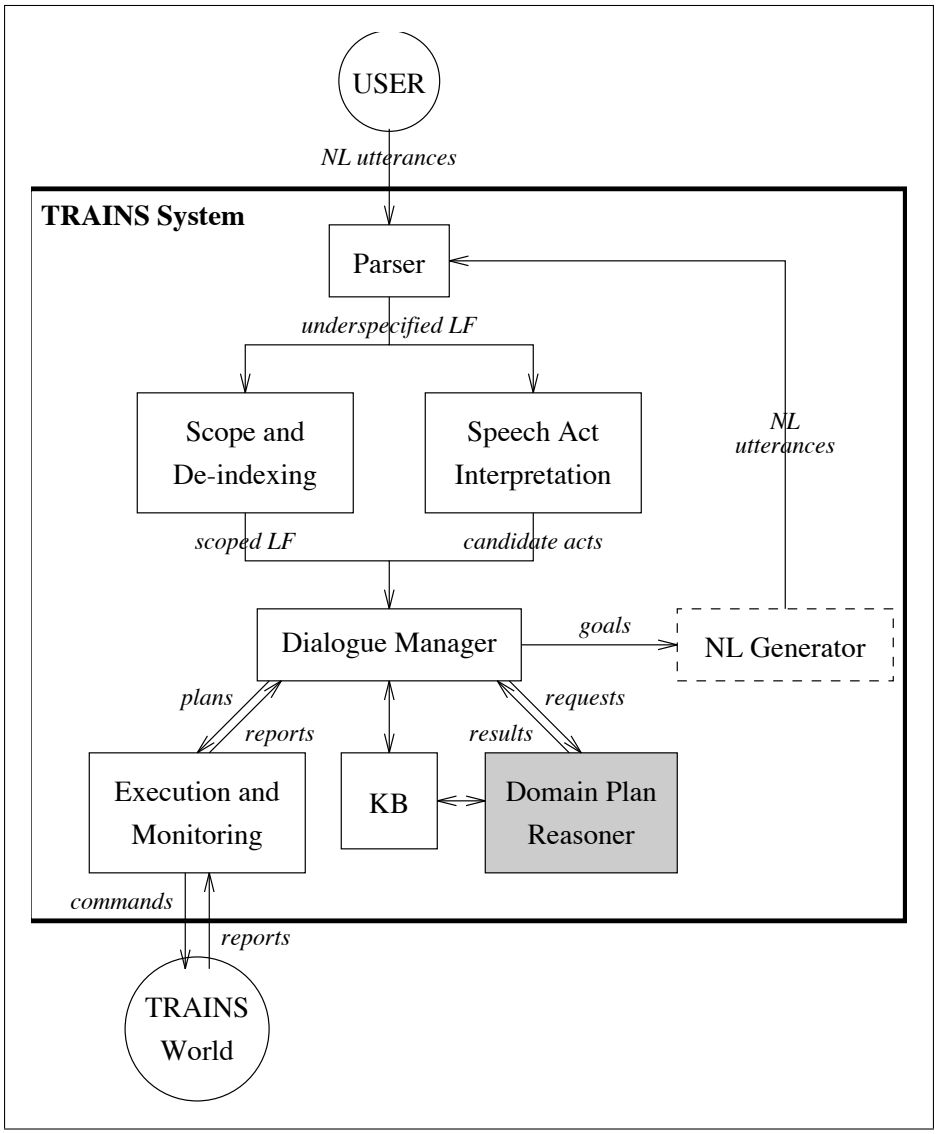


Figure 4.1: The TRAINS system architecture

intensional predicates, tense and aspect, surface speech act operators, and explicit situational variables together with operators used to specify truth in a situation. Of particular importance to the TRAINS system is that the output of the parser is *underspecified*, in that scope of operators and referents of referring expressions are still to be determined.

This underspecified logical form representing the content of the utterance is then processed by a scope and dereferencing module that resolves contextually-dependent expressions (Poesio, 1993b,a). This involves scoping operators to their proper position and resolving anaphoric references based on discourse context. The approach is based on a DRT-like representation named Conversation Representation Theory (CRT) that extends Episodic Logic to better describe the context of interpretation of an utterance (Poesio, 1994). According to CRT, what takes place after an utterance is produced is best seen as a process during which alternative hypotheses about the change in the *discourse situation* brought about by that utterance are obtained. In the TRAINS-93 system, these hypotheses are combined and filtered to produce a final Episodic Logic logical form in which all operators are properly scoped and definite references substituted.

At this point, the representation undergoes conversion from the Episodic Logic representation to one based on the event-based temporal logic described in Chapter 2. The reasons for the translation are twofold. First, as a pragmatic consideration, the work on natural language translation and the work on dialogue agents and plan reasoning were going on simultaneously. Rather than spend time debating a common representation language for all modules, it was faster to allow independent development connected via a syntactic translator. Since both Episodic Logic and the event-based temporal logic are highly expressive languages, the translation need not lose much information required for the interactive planning task. This leads to the second reason for the separation: the idea that the different components have different representational needs. It is not clear that a single representation would be most appropriate for both natural language processing and plan reasoning. For example, Episodic Logic uses a syntax in which argument order reflects surface order in the input text, since this can be an important consideration in determining its meaning (*e.g.*, topicalization, passive voice, *etc.*). These considerations are not as relevant to planning, whereas, for example, a uniform representation of the roles of events and actions and an appropriate temporal representation such as provided by the temporal logic are necessary in order to use planning and plan recognition techniques. Further discussion of the use of multiple knowledge representations in the TRAINS system can be found in (Poesio et al., 1994).

In any case, the Episodic Logic logical form is converted to a speech act representation based on events, which then undergoes Conversation Act analysis to determine the roles the utterance may play in the conversation (Traum and Hinkelman, 1992). The set of candidate acts is then passed to the dialogue manager that determines which of them are appropriate given the current context and acts upon those (Traum, 1994; Traum and Allen, 1994). The dialogue manager maintains belief contexts for the system and manager reflecting its knowledge of the state of the dialogue. It also maintains sets of goals, intentions, and obligations, which form the basis for a reactive model of system

behaviour. For example, the manager asking a question imposes an obligation on the system to address the question, either answering it or rejecting it, or whatever. Once it has decided to answer, the manager adopts the intention of answering which, after checking its knowledge base, leads to the goal of informing the user of the answer.

Among the operations the dialogue manager can perform are calls to the domain plan reasoner. For example, in the case of a suggestion to perform some action, the dialogue manager can ask the plan reasoner to attempt to incorporate the suggested action into the current plan. If successful, the dialogue manager can then accept the suggestion and use information returned from the plan reasoner to generate suggestions of its own. If the incorporation fails for some reason, the dialogue manager will initiate a clarification subdialogue, possibly using information returned by the plan reasoner from the failed incorporation attempt. The dialogue manager can also interact with the generation component (currently only partially implemented) to generate utterances and with the execution component to dispatch completed plans for execution and monitoring (Martin, 1993).

One final thing to note about the system architecture is that the system's utterances are themselves processed by the entire system (with the speaker-hearer modality reversed, of course). This is a principled way of determining the meaning of what was said, so that the system does not mistakenly think that its utterances have (only) their intended effects.

4.2 Plan Reasoner Requirements

From consideration of the overall TRAINS system, I now focus in on the domain plan reasoner module, shaded in Figure 4.1. This section outlines the functionality required of the module as a result of its operating within a larger, language-oriented system. I want to emphasize that these issues are not simply a consequence of the design of the TRAINS system. Rather, I believe, they highlight issues that are traditionally ignored by planners that operate in more restricted environments and that are necessary in any interactive, realistic system.

The first requirement is that the input to the plan reasoner is "language-like." That is, it is expressed in a highly-expressive language with multiple levels of representation. These include the domain level (trains, commodities, cities, time, *etc.*), the plan level (plans, actions, events, enablement, generation, *etc.*), and the problem-solving level (strategies, focus of attention, choicepoints, *etc.*). All of these and more are candidates for discussion in a system such as TRAINS. In all cases, the objects involved may be only partially or indefinitely described, and part of the job of the plan reasoner may be to determine more fully which objects are involved (*e.g.*, "move a boxcar").

The next requirement is that planning and plan recognition (and other plan reasoning) is interleaved during a dialogue. In the TRAINS system, it is the discourse manager that determines what plan reasoner operations are invoked and when, but the phenomenon is more general than this. Plan recognition is typically done to understand the meaning of what has been said, then planning is done to refine the proposal and

possibly generate new suggestions, and plan evaluation is, in a sense, being done continually. Further, the different forms of plan reasoning are performed in different ways. Plan recognition is typically a satisficing process, since we are primarily interested in ensuring coherence of the dialogue. That is, so long as we can make something of the user's statements, we we shouldn't expend too many resources trying to optimize the results. This is because the user is likely to either elaborate on her proposal or modify it, so the results of further reasoning are likely to be unnecessary at best. If we can't figure out what was meant realtively easily, we have probably misunderstood. Planning, on the other hand, is more of an optimizing process, performed when the system has an opportunity to think about the plan. In the case of an intelligent planning assistant, of course, the results are expected to be both accurate and helpful.

This interleaved nature leads to the third requirement, which is that plan reasoning be able to be performed in arbitrary knowledge contexts. We have already noted that the dialogue manager maintains belief contexts representing different views of the conversation. The plan reasoner can be called upon to reason in any of these—in a system-related context the system is "thinking for itself," and in a manager-related context the system is simulating the thought processes of the manager. Because the contents of the contexts depend on what has already been said, the plan reasoner must be able to start with a partial plan in a context and reason from it.

Fourth, the plan reasoner must return useful information. This is not a staggering observation, but the fact that the plan reasoner is just one module of many in the system makes it important and different from traditional systems. The results of plan reasoning operations need to be expressed declaratively so that they can be used by the dialogue manager to generate utterances, or update beliefs, or whatever. Since not everything can be communicated to the user at once, some parts of the information returned by the plan reasoner may not become part of the plan, at least not immediately. And if a plan reasoning operation fails for some reason, it is essential that useful information describing the failure be returned in order that the system can at least inform the user, if not address the problem at the meta-level.

Finally, as we concluded from our analysis of mixed-initiative planning in Section 3.1, much of the reasoning performed by the plan reasoner is defeasible. Indeed, many of the modules in the system are based on a hypothetical reasoning model. The parser, for example, can generate multiple candidate parses and the scope and reference module explicitly reasons about hypotheses regarding the discourse state. For the plan reasoner, this means making assumptions explicit, being able to assert and retract them, and, importantly, realizing that it shouldn't always perform complete reasoning. Rather, it can make an assumption or, since the system is interactive, indicate that it would be better to ask a question.

These five points are by no means exhaustive, but they already impose significant requirements on the implementation. As with the theories described in previous chapters, existing implemented planning and plan recognition techniques are going to be of limited benefit in such a system. The next two sections describe the implementation of representations adequate to fulfill these requirements. Those less interested in the programming details can skip to Section 4.5 regarding the plan reasoning algorithms or

to Section 4.6 that presents the TRAINS-93 sample dialogue.

4.3 Knowledge Representation

Having presented general motivating requirements for the plan reasoner, we can now get into the details of the implementation. This section describes the syntax of EBTL, our implementation of the event-based temporal logic from Chapter 2. EBTL is implemented in Common Lisp on top of the Rhet knowledge representation system(Allen and Miller, 1989). Although Rhet provides a complete logic programming environment, for several reasons EBTL is implemented as a Lisp toolkit rather than a theorem-proving system. After describing the syntax of the language and its implementation, we then describe the various specialized reasoners used by the plan reasoner and their interfaces. We conclude this section with a discussion of the relationship between EBTL and Rhet and of issues for future development of knowledge representations suitable for building systems like TRAINS.

4.3.1 EBTL Syntax

The EBTL language includes terms and variables, both of which can be typed, and formulas that can contain other (quoted) formulas. Several of the syntactic constructs are understood by the Lisp reader as well being accessible through constructor functions. All functions mentioned in this section are described more fully in Appendix A.

Terms: Terms are enclosed in square brackets. For example, `[George]` is a constant term, and `[mother [father [George]]]` is function term. For readability, constant terms inside function terms can omit their brackets, for example `[parent George]` is acceptable.

Types: Ground terms (and variables, see below) can be typed. Types are represented as Lisp atoms and which are organized into a type hierarchy. Unification in EBTL takes account of the types of the objects, forbidding unification between types known to be disjoint and restricting a more general type to a less general one when the two are unified. Typenames are generally prefixed with "`T-`", as in `T-Engine`.

There are two classes of types in EBTL: simple and structured. Simple types are used in unification and are otherwise not special. Structured types, in addition to their unificational properties, define roles, constraints, and additional attributes that instances of the type possess. In the TRAINS system, structured types are used to define events, as will be described in the next section.

Variables: Variables in EBTL are introduced by a question mark, as in `?var`. The variable name can be followed by an asterisk and a type name, as in `?var*type`. Variables participate in unification respecting their types, as expected.

Formulas: Well-formed formulas in EBTL are simply Lisp lists where the `car` of the list is a keyword representing the predicate name and the `cdr` of the list is the arguments, which may include other formulas as well as terms and variables.

A variety of logical operators on formulas are defined that construct formulas from other formulas. These include negation (`:not`), conjunction (`:and`), disjunction (`:or`), and exclusive-or (`:xor`). Each takes an unlimited number of formulas as arguments. A special form exists for specifying restricted existential quantification:

(`:lf-exists` *var dm restrict wff*)

where *var* appears in the restriction *restrict* and the main formula *wff*. For example, the expression "there is a ton of oranges at Avon" would be translated as

```
(:lf-exists ?O*T-Oranges [DM] (:Quantity ?O [1 ton]) (:At ?O [Avon]
                                [NOW]))
```

The *dm* argument is a discourse marker, used to track reference to indefinite objects across utterances. That is, whenever the deindexing module creates an object in the discourse history, it represents it with a discourse marker. When the referent is ultimately identified, it is asserted to be equal to the marker, thereby resolving other references to the same object.

Lambda abstraction: Indefinite terms can be denoted with the following special form:

(`:lambda` *var constraints*)

where *var* is a variable used in the formula *constraints*. This denotes a type whose instances satisfy the constraints. For example, the expression "a boxcar at Corning" would be translated by the statement:

```
(:lambda ?C*T-Boxcar (:At ?C [Corning] [Now])) .
```

The special predicate `:apply` takes a lambda expression and a term as arguments and yields a proposition that is true if the constraints of the lambda expression are true of the term. Lambda expressions are used extensively by the TRAINS plan reasoner, both in its input arising from natural language use of indefinite reference, and to delay Skolemization during reasoning. They play an important role in the assumption-based unification to be described in Section 4.5.1.

4.3.2 Specialized Reasoners

EBTL, like the Rhet system it is based upon, depends on a variety of specialized reasoners to perform crucial inferences. EBTL provides functions to access each of the reasoners, as well as integrating them into the general deductive database mechanism. Again, for more details regarding these functions, see Appendix A.

Knowledge Base: EBTL bases its inferences on the current knowledge base, implemented using Rhet's belief context mechanism. However, belief contexts are not explicit in EBTL—all inference is performed in the current context, whatever that may

be. Typically it will be set by the dialogue manager to reflect the modality in which interpretation of an utterance is taking place. Functions are provided to assert propositions to the knowledge base, to attempt to prove them, and to test consistency of them. Propositions corresponding to the other specialized reasoners described below are treated properly by both the assertion and proof functions, and call the appropriate specialized reasoner.

Types: The special predicate `:type` sets an object's type when asserted and queries it when proven. For example, asserting

```
(:type [Car7] T-Boxcar)
```

would set the type `[Car7]` to be a boxcar.

As noted above regarding lambda abstraction, the special predicate `:apply` can be used to assert or prove whether a type specified by a lambda expression is true of a term. For example, the formula

```
(:apply (:lambda ?C*T-Boxcar (:At ?C [Corning] [NOW])) [Car7])
```

would only be true if `[Car7]` was currently at Corning.

Equality: EBTL supports full equality and inequality for ground terms. The special predicates `:eq` and `:ineq` are used in formulas, for example, asserting

```
(:eq [David] [father George])
```

would add the equality, and querying it would test the equality.

Time: Temporal reasoning in EBTL is supported using the Timelogic interval temporal reasoner (Koomen, 1989) and terms and variables of type `T-Time`. The special predicate `:time-reln` is used in formulas. The interval relations are referred to using keywords (*e.g.*, `:b` for $Before$) or a list of keywords (representing a disjunction). For example, the formula

```
(:time-reln [7AM] :d [Morning])
```

states that 7:00AM is during the morning, and

```
(:time-reln [Morning] (:b :m :mi :a) [Evening])
```

states that the two intervals are disjoint.

Events: Events in EBTL are implemented using Rhet's structured types. These are a very general mechanism provided by Rhet for defining frame-like logical objects. In EBTL, they are used in a very specific way, described here.

Each event type defines a set of *roles*: functions that can be applied to instances of the type to denote individuals participating in the event. These functions can specify a type for the role, and are inherited by subclasses (which can specialize role types).

Rolenames start with the prefix "`r-`"; role functions start with the prefix "`f-`." For example, a `T-Move-Car` event might have a role `r-car` of type `T-Car` denoting the car being moved. The subtype `T-Move-Boxcar` specializes the type of this role to `T-Boxcar`. Note that the roles of an event can be domain objects, times, or even other events. Roles can be referred to either functionally, as in `[f-car E123]` or via the `:role` special predicate. For example,

```
(:role [E123] :r-car [C2])
```

is equivalent to

```
(:eq [f-car E123] [C2]).
```

Each event type then defines a set of *constraints*: formulas that are asserted each time an instance of the type is created. If it is inconsistent to assert them, creation of the instance fails. The constraints cannot currently be accessed in EBTL. They are automatically assserted when `type-skolemize` is called to create a new object of a given type.

Constraints are used for three purposes. First, they define the temporal structure of the event as described in Section 2.2.3 by asserting the required temporal relations between temporal roles of the event. Second, they allow roles to be "synonyms" (*i.e.*, necessarily refer to the same object), by asserting an equality between them. This is useful since the rolenames often come from natural language where, for example, one can refer to either "the object," "the car," or "the boxcar" being moved in a `T-Move-Boxcar` event. Finally, the constraints specify relationships between the event and any sub-events. For example, a `T-Move-Car` event is generated by a `T-Move-Engine` event (under condition of their being coupled, see below). The latter is denoted by the `f-move-engine` role function which denotes an event of type `T-Move-Boxcar` which is constrained to occur simultaneously with the moving of the car, to involve the same car, the same locations, and so on.

The constraints specify all the static properties of an event—those that are a consequence of its existence. Other properties are used in reasoning about how to bring about events and what the effects of their occurrence are. These include effects, enablers (traditional preconditions, temporally qualified), and generators (events that cause the event). Typically a generator generates an event only if the enablers are true. Finer distinctions would be possible but have not been necessary yet in TRAINS. EBTL provides a set of Lisp functions that return information about these relations:

```
enablers-of-event-type     enablers-of-event
effects-of-event-type      effects-of-event
generators-of-event-type   generators-of-event
```

The `-event-type` forms return formulas with a variable `?self` for the event; the `-event` forms substitute the given event. Finally, there is the function

```
achievers-of-fact
```

that returns a list of lambda expressions denoting events with an effect that unifies with the fact (sort of the inverse of `effects-of-event-type` with unification assumptions). These functions are all used by the plan reasoning algorithms as they explore the space of plans.

4.3.3 Relationship to Rhet

Before moving on to the description of the representation of plans and the plan reasoning algorithms, we want to comment briefly on the relationship between EBTL and Rhet. As noted at the outset, EBTL is built on top of Rhet, and includes many of the important features of Rhet, such as the various specialized reasoners. Why then did we go to the trouble of defining a separate specification? Readers unfamiliar with Rhet and uninterested in the evolution of a complicated knowledge representation system can skip this section.

The pragmatic reason for the separate specification of EBTL, as with most of the TRAINS system, was that Rhet was itself evolving simultaneously with the development of the plan reasoner. It therefore seemed appropriate to specify an interface language, EBTL, in order to minimize the work required should Rhet change in significiant ways (which it certainly has done over the course of the project).

Second, as Rhet has evolved, it has grown to provide significantly more features and functions than needed for reasoning using EBTL. The motivation behind EBTL then was to provide a uniform, compact, and intuitive language that precisely characterized the needs of the domain plan reasoner. This specification could then serve as a target for future development of Rhet (and its successor, Shocker, mentioned below).

An example of functionality that, while powerful, wasn't directly necessary is Rhet's belief context mechanism, in particular the peculiar modal operators it adds to the language. The dialogue manager uses belief contexts to represent the state of knowledge of the dialogue participants. Inheritance from parent contexts is certainly desirable (*e.g.*, from the shared context), and is provided by Rhet. However, not only do Rhet's belief contexts clutter the syntax of the language, they don't provide (easily) several of the features one might want, such the ability to compare the contents of two contexts, to distinguish the literal contents of the context from their implications, and an appropriate interaction with the specialized reasoners. In any case, the plan reasoner needn't know about contexts at all. It is told to reason in a particular context, and returns information based on the knowledge within it. Thus contexts are not a part of EBTL, although they or something like them are necessary in the overall system.

In addition to these software engineering issues, there were some serious shortcomings of Rhet that needed to be addressed in order to implement the plan reasoner. One of these was the inability of Rhet to represent meta-statements appropriately. Since the representation of plans labels nodes of the plan graph with formulas (described in the next section), it was essential that EBTL be able to represent statements such as:

```
(:plan-enabler (:in [O1] [CAR1] [NOW]) [E123])
```

i.e., the oranges being in the boxcar enables event `[E123]`. This led to a fairly convoluted implementation of the EBTL knowledge base functions in order to work around the problem.

Other problems were due to the original emphasis on Rhet as a logic programming environment. Because of this, the Lisp interface had evolved somewhat haphazardly, with some functions being inconsistent in their use of arguments, and with others being forced into use in a context in which they were not designed to operate, resulting in run-time errors. The multiple internal datatypes, while hidden from the logic programmer, were difficult to work with in Lisp. Unfortunately, we could not implement the plan reasoner algorithms in the logic programming environment, for reasons of expressivity, efficiency, and control. EBTL was therefore designed as a Lisp toolkit from the outset, with no explicit support for logic programming beyond prove/assert, leaving that to Rhet if it was needed.

In the final analysis, Rhet provides a powerful, if somewhat obscure set of tools (*e.g.*, types, equality, time, *etc.*) for building sophisticated reasoning systems. EBTL provides a clearer interface to precisely those elements required by the plan reasoner, and serves as an example of the functionality required by a plan reasoning system that functions in a realistic and complex environment. The successor to Rhet, Shocker, addresses several of the shortcomings described in this section, and will provide a solid basis for future generations of the TRAINS system. Since the domain plan reasoner module is based on EBTL, we expect a straightforward migration from Rhet to Shocker. Work has already begun on using the EBTL specification to define a Shocker module for use by the plan reasoner, a considerably simpler task than porting the entire Rhet interface to Shocker.

4.4 Plan Representation

EBTL is the basis on which the representation of plans is built. Not only do elements of plans refer to EBTL sentences, EBTL is itself used for the declarative representation of plans using meta-predicates. This section first describes the data structures used to represent the plans based on the plans-as-arguments formalism described in Chapter 3. EBTL predicates that form the declarative plan description language used in communication are described next, followed by the basic plan space searching functions that underlie the plan reasoning algorithms described in the next section. Complete details of all functions and data structures is available in Appendix B.

4.4.1 Plan Graphs

Recall from Section 3.2.1 that an argument is a connected, directed acyclic graph whose nodes are labelled by propositions. The internal representation of these used by the TRAINS domain plan reasoner is as Lisp structures formed into DAGs using Lisp lists. This is the most efficient and practical way of performing the necessary operations.

Plan graphs are constructed using nested lists of two types of structures, `plan-node`s and `plan-link`s, corresponding to the propositions and steps in an argument. The nodes

are partitioned into those nodes labelled by events (implying a `:occurs` proposition) and those labelled by other propositions.

For example Figure 4.2 shows a sample plan graph graphically. Event nodes are square; fact nodes are rounded. Assume that `[E1]` denotes an event of type `T-Move-Oranges` with appropriate role bindings, `[O1]` denotes some oranges, `[C1]` denotes a boxcar, and `[Avon]` and `[Bath]` denote cities. Then this is an argument that moving the oranges from Bath to Avon using the car will succeed in their being at Avon, given that both the car and the oranges are now at Bath (the figure ignores temporal parameters).

The representation of this argument as a plan graph involves four `plan-node`s, three labelled by propositions and one labelled by the event. Each of the arrows in the figure corresponds to a `plan-link`, with a dummy link for the root note. The resulting Lisp plan graph is shown pretty-printed in Figure 4.3. Figure 4.4 shows our CLIM display of the same graph. The CLIM plan graph viewer also provides point-and-click inspection of the labels of the nodes and their arguments, and can easily be extended to support editing of the parts of the plan, for example.

This approach to representing plan graphs may seem arcane, but in fact is has several points to recommend it. In addition, remember that we will rarely need to build plan graphs by hand—they are constructed by the plan reasoning algorithms to be described shortly. First, the structure of the `plan-graph` uses standard Lisp list nesting to represent the structure of the graph. This means that it is relatively easy to examine the graph using standard Lisp functions and pretty-printers. An alternative would be to represent the graph as a list of nodes and a list of ordered pairs representing the links. This might make it simpler to construct plan graphs, at a loss of readability (which could be rectified with a pretty-printer). Second, it may seem that the `plan-link` structures are unnecessary, since they could as easily be represented by, say, the type keyword as the `car` of the list representing the subgraph. However, since in fact links have more attributes that just their type (for example, the numbers in the figures above are an `index`, used to remember where the formulas came from), it seemed better to put them in an easily-extendable structure. If the attributes were stored in a list, either they would all have to be present and in a fixed order, or a keyword-value system would have to be used, which amounts to using a structure anyway. Finally, the plan reasoner module was written to use interface functions rather than relying on internal details of the plan graph representation. This ought to make it fairly straightforward to change the representation, as indeed it changed completely once during the course of development of the module.

4.4.2 Plan Description Predicates

The data structures described in the previous section are used internally by the plan reasoner. To communicate with other modules, a language of *plan description predicates* is defined that allows plan graphs to be declaratively represented. The translation between the plan graph data structures and the plan description predicates is quite straightforward.

Corresponding to the `plan-node`s, there are two predicates:

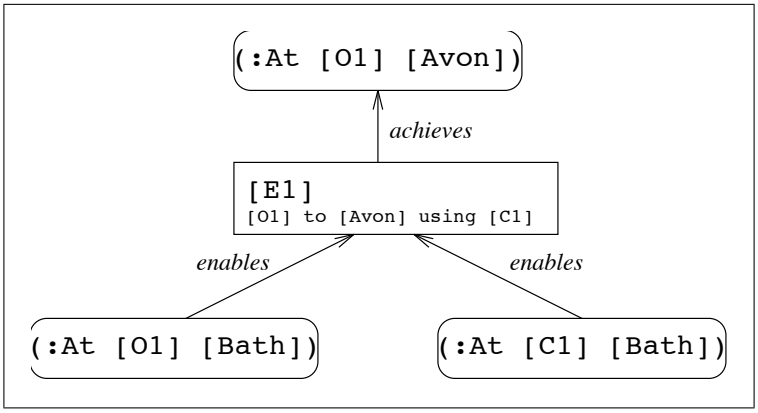


Figure 4.2: Sample plan graph

```
((#NIL0<#FACT<(:AT [O1] [AVON])>>
  (#ACH0<#EVENT<[E1]>>
   (#ENB0<#FACT<(:AT [O1] [BATH])>>)
   (#ENB0<#FACT<(:AT [C1] [BATH])>>))))
```

Figure 4.3: Sample plan graph: Lisp output

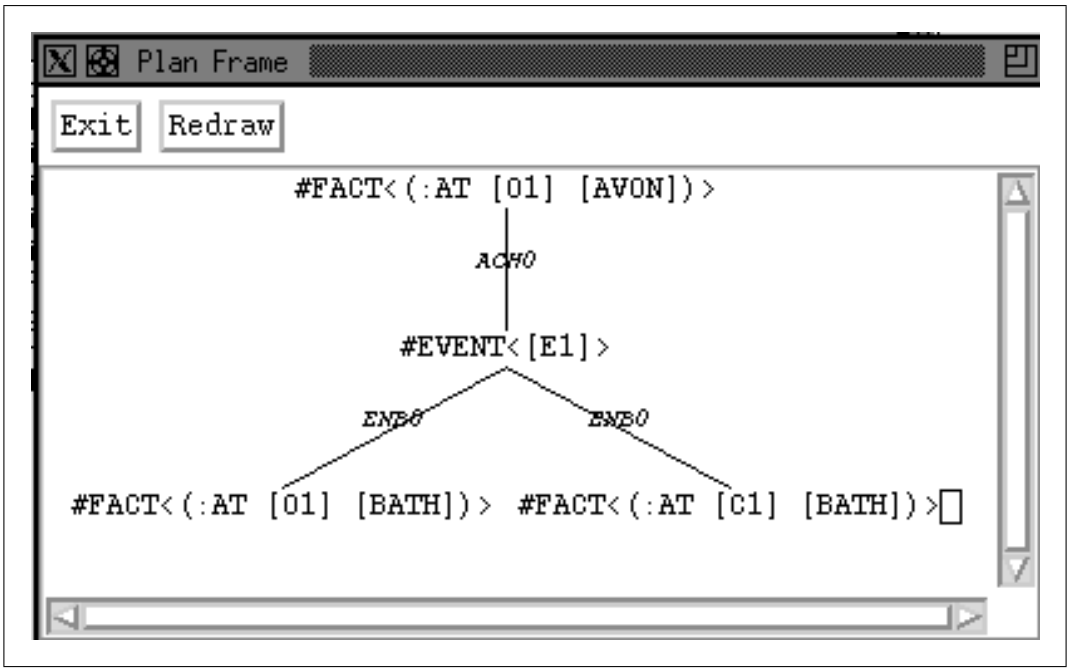


Figure 4.4: Sample plan graph: CLIM display

```
(:plan-event event plan)
(:plan-fact wff plan)
```

These take two arguments: a term denoting a plan and a formula or event term, as appropriate. Each `plan-link` is represented by one of the following predicates:

```
(:plan-enables wff event plan)
(:plan-achieves event wff event)
(:plan-generates event1 event2 plan)
(:plan-supports wff1 wff2 plan)
```

In addition, there are predicates `:plan-goal` and `:plan-premise` that describe the sinks and sources of the graph, respectively. These are redundant in describing any particular plan, however they may be used as constraints on a plan that is only partially specified. In fact, however, the nonmonotonicity of these two predicates poses a problem: what is a premise at one point may not be after further reasoning even if the fact or event itself is still in the plan. Our approach is that the plan reasoner only describes plans as it sees them—management of this changing view of the plan is up to higher-level modules, at least in the current system. The set of plan description predicates describing the example plan graph from Figure 4.2 is shown in Figure 4.5, where the term `[PLAN-1]` denotes the plan.

The idea is that other modules communicate with the plan reasoner by passing a term denoting the plan under consideration as part of each request. The plan reasoner then builds up the plan graph corresponding to the plan description predicates provable in the current belief space. It then reasons with the plan graph, and converts the results back into plan description predicates for use by the calling module. That module may decide to assert some or all of this information, thereby augmenting the plan, in some, possibly different, belief space, or use the information for other purposes, for example generation of utterances. The declarative representation using plan description predicates allows other modules to determine the context in which the plan reasoner operates, which is one of the design goals noted in Section 4.2. It would be extremely awkward and computationally expensive to have the plan reasoner work directly with the declarative representation.

4.4.3 Plan Graph Search Procedures

Many of the plan reasoning algorithms described in the next section search the space of plan graphs defined by the event definitions. That is, the event definitions define a space of plan graphs where each link in a graph corresponds to an enabler (precondition), effect, or generator in some event definition. To facilitate this, the function `plan-graph-expand` takes as input a plan graph and returns a new plan graph where each leaf node of the original has been expanded one level. If the leaf was an event node, the new leaves will be labelled by enablers and generators. If it was a fact node, they will be labelled by lambda expressions denoting events that could achieve the fact labelling the node. These may be Skolemized during further reasoning, becoming

```
(:PLAN-FACT (:AT [O1] [AVON]) [PLAN-1])
(:PLAN-EVENT [E1] [PLAN-1])
(:PLAN-FACT (:AT [O1] [BATH]) [PLAN-1])
(:PLAN-FACT (:AT [C1] [BATH]) [PLAN-1])
(:PLAN-ACHIEVES [E1] (:AT [O1] [AVON]) [PLAN-1])
(:PLAN-ENABLES (:AT [O1] [BATH]) [E1] [PLAN-1])
(:PLAN-ENABLES (:AT [C1] [BATH]) [E1] [PLAN-1])
(:PLAN-GOAL (:AT [O1] [AVON]) [PLAN-1])
(:PLAN-PREMISE (:AT [O1] [BATH]) [PLAN-1])
(:PLAN-PREMISE (:AT [C1] [BATH]) [PLAN-1])
```

Figure 4.5: Sample plan graph: Plan description predicates

events in the plan (the events are not created immediately simply to avoid performing that expensive operation unnecessarily).

There are two issues to consider here, one dealing with bookkeeping, one conceptual. First, this is where the `index` field of the `plan-link` structure comes into play. This field is used to record from what "slot" in the event definition the label of the link came. During expansion, a link is not added if a link with the same index is already present. This is significant because, as will be described shortly, the plan reasoning algorithms can generate plan graphs with long branches which must be searched for siblings but not duplicated.

The more fundamental issue raised by this approach is the fact that it will not generate arbitrary fact-fact (supports) links justified by the underlying inference engine. While this could easily be added, at least for certain syntactically-recognizable constructions, much of this functionality is already provided by the assumption based unificiation procedure to be described in the next section. For example, suppose we know `(:At [O1] [Avon] [T1])` and we are attempting to unify it with `(:At [O1] [Avon] [T2])`. This would normally require a persistence assumption, which could be represented by a supports link. However, the assumption-based unifier will allow us to do this by assuming that `[T1]` includes `[T2]`, if consistent. Examples of this kind of fact-fact link will be shown in the sample dialogue in Section 4.6. On the other hand, one can think of many other instances where a specialized reasoner could be used to generate supports links, for example reasoning about containment (`:In`) and location (`:At`). Exactly what the right approach is remains a subject for experimentation.

Finally then, the iteratively-deepening breadth-first search of the space of plan graphs is performed by the function `plan-graph-search`. It takes a plan graph and a termination test and repeatedly expands the plans and applies the test until either the test returns non-`NIL` or a global depth bound is exceeded. All the plan reasoning algorithms described in the next section use `plan-graph-search` with an appropriate termination condition that returns information about the result of the operation, whether it was successful or not.

4.5 Plan Reasoning

In this section we describe the graph-based algorithms used by the domain plan reasoner to provide the services outlined in Sections 4.1 and 4.2. We start by considering the assumption-based nature of this reasoning and describing our assumption-based unification technique that underlies all the plan reasoning. We then describe the plan recognition component that allows *incorporation* of several types of information into an existing, arbitrary, plan graph. All forms of incorporation are based on the basic iteratively-deepening breadth-first search procedure that results in a "shortest-path" incorporation preference. We believe that this satisficing approach is appropriate for plan recognition in an interactive system, since so long as the utterance is coherent, we should wait until the system gets the turn to do more sophisticated reasoning. Finally, we describe the *elaboration* routine that does reasoning to flesh out details of a plan and detect conflicts.

4.5.1 Assumption-Based Unification

We have already argued that much of the reasoning that goes on in mixed-initiative planning is defeasible, that is, based on assumptions and subject to revision given new information or more time to reason (*cf.* Section 3.1). It is logical therefore that the heart of the TRAINS domain plan reasoner is an assumption-based procedure, `plan-unify`. This function takes two formulas and attempts to unify them, returning an indication of success or failure and, if successful, a set of assumptions necessary for the unification. A pseudo-code description of the relevant functions is shown in Figures 4.6 and 4.7.

In unifying two formulas, if the predicate names are the same, `plan-unify` compares arguments pairwise and attempts to unify them, as one would expect. For non-event terms or variables, this amounts to testing equality. Two events are equal if their roles are all equal, since events are uniquely determined by their roles (*cf.* Section 2.2.3) and this is enforced by the EBTL implementation. If the terms are known to be equal, unification succeeds and no assumptions are needed. If they are known to be unequal or if an assertion of their equality would be inconsistent in the current context, unification fails. Otherwise `plan-unify` succeeds and returns the equality assumptions necessary for unification. Note that, of course, all assumptions must be mutually consistent, not just individually so.

The unification procedure is complicated by the possibility that one or the other object being tested may be a lambda expression. For example, the plan reasoner might be attempting to unify an expression

```
(:lambda ?X*T-Boxcar (:At ?X [Avon] [Now]))
```

coming from the utterance "a boxcar at Avon" with a particular boxcar, say `[C1]`. In this case, if the constraints of the lambda expression can be consistently asserted after substituting the term, then the substituted formula is returned as the assumption supporting the unification.

In the case of unifying two lambda expressions, I believe that the correct approach would be to Skolemize one of them and then attempt to unify it with the other. The result, if successful, would be the conjunction of the two constraints together with an equality between the variables. However, neither EBTL nor Rhet can reason effectively with equalities involving non-ground terms, so it is not clear whether this result could be used. Since this did not arise in the plan reasoner to date, we did not pursue it, but it is something that needs to be worked out as the EBTL language evolves.

To summarize, `plan-unify` relies on the underlying reasoner (in particular the equality subsystem) to test consistency of proposed unifications, and returns the consistent ones for use by the higher-level plan reasoning algorithms.

4.5.2 Incorporation

As noted in Section 4.2, one of the principal requirements of the plan reasoner was the ability to support interleaved planning and plan recognition. Of these, the plan

```
plan-unify (wff1 wff2)
  if (LISP-EQUAL wff1 wff2)
  then return (values t nil)
  else if predicate name and arity are the same
  then return (plan-unify-args (wff-args wff1) (wff-args wff2))
  else return (values nil nil)
  endif

plan-unify-args (args1 args2)
  results := nil
  for i := 1 to (length args1) do
       (flag,assmpts) := (plan-unify-object args1[i] args2[i])
       if flag
       then results := results + assmpts       ; ok:  gather assmpts
       else return (values nil nil)            ; fail
       endif
  done
  if (kr-consistent-p results)
  then return (values t results)               ; ok:  all assmpts
  else return (values nil nil)                 ; fail
```

Figure 4.6: Pseudo-code description of `plan-unify`

```
plan-unify-object (obj1 obj2)
  if (LISP-EQUAL obj1 obj2)
  then return (values t nil)                   ; ok:  already identical
>else if both objects are lambda expressions
  then error                                   ; see text
  else if one object is (:lambda lvar lwff) and the other is obj
  then assmpt := replace lvar by obj in lwff
  else assmpt := (:eq obj1 obj2)
  endif
  if (kr-prove assmpt)
  then return (values t nil)                   ; ok:  no assumptions
  else if (kr-consistent assmpt)
  then return (values t (assmpt))              ; ok:  new assumptions
  else return (values nil nil)                 ; fail
  endif
```

Figure 4.7: Pseudo-code description of `plan-unify-object`

recognition function, `incorporate`, is the most interesting. A pseudo-code description is shown in Figures 4.8 and 4.9.

In its simplest form, `incorporate` takes three arguments:

- a formula or term to incorporate;
- a term denoting the plan into which the object should be incorporated; and
- a flag indicating the type of incorporation (*e.g.*, `:goal`).

It returns a list consisting of

- A flag, indicating success (`:ok`), ambiguity (`:choice`), *etc.*;
- a list of plan description predicates describing elements added to the plan in the course of incorporation;
- the resulting `plan-graph`; and
- a *context* node, usually the node corresponding to the input item.

The `incorporate` function first converts the plan term into a plan graph (not shown in the pseudo-code). For efficiency, since this is an expensive operation involving gathering up instances of the various plan description predicates provable in the current knowledge base, the plan graph can be maintained by the dialogue manager and passed as a keyword argument instead.

Ignoring for now the context and purpose processing indicated in Figure 4.8, the formula being incorporated is tested syntactically with the result that either an event, object, or enabler is incorporated. Incorporating an enabler amounts to incorporating some event that is enabled by the fact. In the other two cases, the plan graph is first searched for a node whose label unifies with the object being incorporated. If this is successful, only the assumptions needed for the unification are returned. If there are several such nodes, a `:choice` is returned together with the sets of relevant assumptions.

Otherwise, the graph is expanded breadth-first until a unifying node is found (or until a depth bound is reached). If such a node is found, then, in addition to the assumptions required for the unification, plan description predicates corresponding to the path from the successful node to the original graph are returned as well. Again, if several such nodes are found at the same depth, a `:choice` is returned together with the possible paths. If the depth bound is reached, an indication of this is returned, together with the current graph. It is intended that the calling module could pass this graph back to `incorporate` to pick up where it left off, if the situation warranted.

In addition to a success or failure indication and the assumptions and additions to the plan, `incorporate` returns a node corresponding to the point in the graph where the unification finally succeeded (not shown in the pseudo-code). We call this the *context* node, since it can be used by the higher-level modules to focus the search on subsequent utterances to an appropriate subgraph of the plan. This is done by passing it back as the `:context` argument to `incorporate`, as shown in Figure 4.8.

```
incorporate-in-graph (wff graph &key incorp-type context purpose ...)
  if context
  then find subgraph of graph rooted at context
       call incorporate-in-graph with that as graph
  else if purpose
  then call incorporate-in-graph to incorporate purpose
       if unambiguously successful
       then call incorporate-in-graph to incorporate wff in the subgraph
              rooted at the focus returned from the purpose call
       else if multiple choices from purpose call
       then try each choice as above and hope all but one fail completely
       else fail                                ; presupposition failure?
       endif
  else call (incorp-fact wff graph ...)
  endif
  if incorporation failed and incorp-type is :goal
  then return appropriate new graph
  else return results of incorporation
```

Figure 4.8: Pseudo-code description of `incorporate`

```
incorporate-fact (fact graph &key ...)
  if fact is (:lf-exists var dm constraint (:occurs var))
  then call (incorporate-event (:lambda var constraint) graph ...)
  else if fact is (:lf-exists var dm constraint wff)
  then substitute dm for var in wff
       if consistent, call (incorporate-fact wff graph ...)
  elseif fact is (:use-object obj)
  then call (incorporate-object obj graph ...)
  else call (incorporate-enabler fact graph ...)
  endif

incorporate-event (event graph &key ...)
  if event unifies with the label of an event node in graph
  then return unifying assumptions as :plan-supports link
  else search-plan-graph until such an event node is found
  endif

incorporate-object (obj graph &key ...)
  if obj unifies with a role of an event labelling an event node in graph
  then return unifying assumptions as :plan-supports link
  else search-plan-graph until such an event node is found
  endif

incorporate-enabler (wff graph &key ...)
  for each event in (enablees-of-fact wff) do   ; event will be a lexpr
       call (incorporate-event event graph ...)
  done
```

Figure 4.9: Pseudo-code description of `incorporate-fact`, *etc.*

The `incorporate` function also accepts a `:focus` argument that indicates a particular object on which the utterance is focused, as determined by the language modules. For example, in "There are oranges at Corning," the oranges are the focus, not Corning. Although currently unused, this might be useful in disambiguating between possibilities on the basis of their relation to the focus object.

Two comments are in order about this procedure before considering some of the additional functionality of `incorporate`. First, because the graph is searched using `plan-graph-search` (Section 4.4.3) which uses `plan-graph-expand`, it will only consider graphs that are well-formed with respect to the event definitions. That is, while there is nothing to prevent the creation of non-conforming plan graphs, `incorporate` will never create one itself. How does this relate to the claim that is important to be able to represent and reason about faulty plans? In the first place, they can be represented, just not recognized. Second, the "true" event descriptions could be augmented by a "bug library" that was also considered as the plan graph was being expanded during incorporation (but not, of course, during planning). And finally, even if we cannot recognize faulty plans directly, higher-level modules can use the information returned from the unsuccessful incorporation to attempt to resolve the problem. This might result in a clarification sub-dialogue that led to either a revision of the system's event definitions or a modification of the manager's proposal. In this way we exploit the interactivity of the system to offset its limitations, as noted in Section 4.2.

Goals

Special treatment is appropriate for certain forms of incorporation. In the TRAINS system, the recognition of goals is not a function of the plan reasoner, at least not directly (as it is in traditional plan recognition). Rather, linguistic cues such as a modality like "should" or "must" indicate a possible goal of the plan. This is communicated to the plan reasoner via the `:incorp-type` parameter to `incorporate`.

Currently, the only special processing done for goals is that if incorporation would otherwise fail, a new plan graph rooted by the object being incorporated is created and plan description predicates describing it are returned. Otherwise the node ultimately found by the search is simply marked as a goal for future use, as well as being returned as the context for future searches. This strategy is particularly appropriate in our dialogues where the manager is almost required to start by identifying a goal at the outset.

Note that this is not quite an accurate reflection of the plans-as-arguments formalism, where the goals of a plan are exactly the conclusions of the plan-as-argument, that is, the sinks of the plan graph. By allowing non-root nodes to be marked as goals, we are glossing over an important issue, namely the relationship between subplans and their goals and the goals of the overall plan. Since the subgoal is a goal of its subplan, considered as a sink of the subgraph, this is not entirely incorrect. The context mechanism described previously is an attempt to provide a mechanism for referring to subgoals, and in fact in the current TRAINS system, the dialogue manager manipulates these nodes like a "goal stack," backing up the stack if incorporation in a subgraph fails.

But clearly a more detailed analysis of the linguistic phenomena and their relation to discourse state is required before the plan reasoner can be extended to a fuller treatment of goals and subgoals suitable for mixed-initiative planning.

Purpose constructions

Another special case of the general incorporation procedure is provided to handle purpose constructions such as "Use the engine to pick up the boxcar." These constructions are ubiquitous in problem-solving dialogues such as ours. The special handling is invoked by passing another formula as the `:purpose` argument to `incorporate`.

The procedure then attempts to incorporate the purpose first, as shown in Figure 4.8. In the "use the engine" example, that would mean attempting to incorporate picking up (coupling) the boxcar. If this fails or is ambiguous, a special value is returned to distinguish it from normal failure. This is because failure to incorporate the purpose usually implies some kind of presupposition failure, which is usually treated differently in language than incoherence of the main clause. If the incorporation of the purpose succeeds, then the main object is incorporated with the search restricted to the context returned from the purpose incorporation, thus ensuring that it is "for" the purpose. In the example, this means restricting the search to the plan graph rooted at the event of picking up the boxcar and attempting to incorporate use of the engine.

This simple strategy is unlikely to handle all forms of purpose clauses, in particular those involving the mental state of the agent, which is not represented explicitly in the plan graph (for example, "Open the door to see what's inside," although this could be handled by adding perception acts and knowledge preconditions, so perhaps it's not a problem with the plan representation *per se*). In any case, it is effective in reducing the ambiguity that would otherwise result from searching the entire plan graph inappropriately.

In fact, this kind of reasoning could and perhaps should be performed by another component of the system. That is, I believe that a proper treatment of, for example, presupposition failure, would be better obtained if the two-part process described above was carried out by the dialogue manager. Then it would be able to discriminate between failure modes without a need to clutter up the plan reasoner interface (both input and output). It might have more sophisticated language-based techniques for dealing with such failures, techniques that the purely domain-oriented plan reasoner lacks. However, for historical purposes the plan reasoner provides at least this simple mechanism, and it has proven surprisingly effective. Besides, the division between the dialogue manager and the plan reasoner is not always clear and clean.

4.5.3 Elaboration

The other aspect of the domain plan reasoner is a planning component provided by the `elaborate` function. The idea is that while the recognition performed by `incorporate` is intended to be "quick and dirty" in order to keep the conversation going, the elaborate function will be called when the system gets the turn and has some

time to think. Whereas incorporation is a satisficing process, elaboration should be more of an optimizing process, identifying problems missed by incorporation and filling out details of the plan as well as possible.

As currently conceived, `elaborate` takes as argument a term denoting a plan, converts it to a plan graph, and then performs the following operations:

1. Identify unfulfilled preconditions (enablers) and either unify them with a fact already known or in the plan (*i.e.*, persistence, cast in terms of assumption-based unification) or plan to achieve them by adding actions to the plan.

2. Identify actions that cannot be directly executed and decompose them by adding their generators to the plan. For example, a `T-Move-Car` event cannot be brought about directly, but must be generated by a `T-Move-Engine` event when the engine and car are coupled.

3. Identify roles of events in the plan that are not specified and attempt to find objects that satisfy them. In fact, this is typically looked after by the previous two steps.

Thus `elaborate` is a fairly traditional means-ends planner except that it can start with an arbitrarily-specified partial plan rather than simply a goal, and of course everything is temporally explicit.

Not only is the elaboration component simple, it is also stupid. At any point in the process, if it detects an ambiguity, such as a choice of actions that could achieve a goal, it stops and signals the error to the calling module. This is clearly unreasonable, since one of the principal capabilities of the system as a whole is supposed to be the ability to weigh factors like these that the manager may not know about and make intelligent suggestions. Note, however, that many of the choices that a traditional nonlinear planner has to make involve orderings between steps, a problem obviated by our use of the more explicit temporal representation (*cf.* Section 2.1.3).

There are some reasons why the elaboration component is so inadequate. First, in our dialogues significantly more of the interaction involves recognition, to the extent that the plan is fairly complete before the system really gets a chance to elaborate it. In the current system, the dialogue manager determines when to call `elaborate`, based on turn-taking and the discourse state (*cf.* (Traum and Allen, 1994)). An example of this is the TRAINS-93 sample dialogue presented in the next section. This led to a primary emphasis on the development of a flexible and powerful incorporation component to the detriment of the elaboration component.

Second, although traditional planning techniques can be applied to some extent, the fact is that standard non-deterministic approaches are not suitable for interactive systems (*cf.* Section 3.5.2). That is, the choice of whether to persist or plan for a precondition is a trivial non-deterministic operation in all traditional planning algorithms—the planner will backtrack later if the choice was incorrect. But in a system where choices develop the plan and these choices may result in communication and modification of the belief state before all possibilities are considered, the system will be unable to simply

backtrack. A complete analysis of the ramifications of interactivity on planning systems will no doubt be the focus of future work on mixed-initiative planning, and I certainly have not resolved it here.

Finally, much of what makes a planner an "intelligent" assistant is knowledge about the domain, such as resource reasoning, utility tradeoffs, and user preferences. Some preliminary design work has been done on using the statistical reasoning capabilities of the execution module to resolve some choices, such as choosing actions to achieve effects with maximum likelihood. But integration of these modules, and construction of a truly useful elaboration component in general, remains a goal for the next generation of the plan reasoner.

4.6 The TRAINS-93 Sample Dialogue

This section presents the TRAINS-93 sample dialogue, shown in Figure 4.10, with an emphasis on the domain plan reasoner component and its interaction with the dialogue manager. The sample dialogue was a "cleaned-up" version of an actual dialogue recorded with a person playing the role of the system. We removed speech errors and other non-linguistic utterances, and simplified or deleted certain utterances that were expected to be too complex for the current generation of the system to handle. The aim of TRAINS-93 was a system that could participate in the dialogue, given the manager's utterances. As emphasized previously, although this could easily be done with an Eliza-like system, we were interested in well-founded and extensible theories such as those described in the previous chapters that could account for the observed behaviour. All examples in this section are taken literally from a run of the system, and only edited slightly for readability.

The manager starts the dialogue by stating her goal:

```
M:  We better ship a boxcar of oranges to Bath by eight a.m.        (1)
```

This results in the following call to the plan reasoner to incorporate the suggestion:

```
(INCORPORATE
   (:LF-EXISTS ?v4574*T-ORANGES [X3]
    (:QUANTITY ?v4574 (:BOXCAR-UNIT 1))
    (:LF-EXISTS ?L*T-MOVE-COMMODITY [E0]
     (:APPLY
      (:LAMBDA ?E*T-MOVE-COMMODITY
       (:AND (:ROLE ?E :R-AGENT [SYSHUM])
             (:ROLE ?E :R-COMMODITY ?v4574)
             (:ROLE ?E :R-DST [BATH])
             (:TIME-RELN [F-TIME ?E] :B [8AM])))
      ?L)
     (:OCCURS ?L)))
  [PLAN-4719]
  :INCORP-TYPE :GOAL)
```

```
M: We better ship a boxcar of oranges to Bath by eight a.m.          (1)
S: Okay.                                                             (2)
M: So we need to get a boxcar to Corning, where there are oranges.   (3)
M: There are oranges at Corning. Right?                              (4)
S: Right.                                                            (5)
M: So we need an engine to move the boxcar. Right?                   (6)
S: Right.                                                            (7)
M: So there's an engine at Avon. Right?                              (8)
S: Right.                                                            (9)
M: So we should move the engine at Avon, engine E1, to Dansville to (10)
   pick up the boxcar there.
S: Okay.                                                             (11)
M: And move it from Dansville to Corning.                            (12)
M: Load up some oranges into the boxcar.                             (13)
M: And then move it on to Bath.                                      (14)
S: Okay.                                                             (15)
M: How does that sound?                                              (16)
S: That's no problem.                                                (17)
M: Good.                                                             (18)
S: The plan has been fully specified and executed. Goodbye.          (19)
```

Figure 4.10: TRAINS-93 sample dialogue

The formula being incorporated is read as follows. There exist some oranges, referenced by discourse marker [X3] and restricted to being 1 BOXCAR-UNIT in quantity, and there exists an event of type T-MOVE-COMMODITY referenced by discourse marker [E0] and restricted to having agent [SYSHUM] (*i.e.*, "we"), commodity role the oranges, destination Bath, and time before 8:00 a.m., such that that event occurs. This formula is to be incorporated into the plan denoted by [PLAN-4719] (currently empty, of course) and the incorporation is flagged as a goal.

As shown in Figure 4.8, when asked to incorporate occurrence of an event, the plan reasoner attempts to find or add an event node unifying with the input event (here specified as a lambda expression). Since this is not possible starting from an empty plan, the goal handling mechanism of Section 4.5.2 is invoked. The result is that a new event is created, [MV-COMM-4779], the constraints of the lambda expression are asserted of it, and a new event node becomes the only element of the resulting plan graph. The plan description predicates returned are:

```
((:PLAN-GOAL (:OCCURS [MV-COMM-4779]) [PLAN-4719])
 (:PLAN-EVENT [MV-COMM-4779] [PLAN-4719])
 (:PLAN-PREMISE (:OCCURS [MV-COMM-4779]) [PLAN-4719]))
```

That is, a single node has been added to the plan, an event node labelled by the event of moving the oranges to Bath, and it is both a source and a sink of the resulting plan-graph. The context node for subsequent calls is, of course, the newly-created event node:

```
#EVENT-GOAL<[MV-COMM-4779]> .
```

Since the manager has paused, thereby releasing the turn, the system takes the initiative and acknowledges the suggestion:

```
S:  Okay.                                                          (2)
```

In so doing, the dialogue manager performs tasks such as moving the current view of the plan into the shared belief space, and updating its obligations. The manager then starts specifying aspects of the plan:

```
M:  So we need to get a boxcar to Corning, where there are         (3)
    oranges.
```

This results in the following call to the plan reasoner:

```
(INCORPORATE
   (:LF-EXISTS ?v6229*T-BOXCAR [X244] NIL
    (:LF-EXISTS ?L*T-MOVE [E236]
     (:APPLY
      (:LAMBDA ?E*T-MOVE
       (:AND (:ROLE ?E :R-AGENT [SYSHUM])
             (:ROLE ?E :R-OBJECT ?v6229)
             (:ROLE ?E :R-DST [CORNING])))
      ?L)
     (:OCCURS ?L)))
```

```
[PLAN-4719]
:INCORP-TYPE :GOAL
:CONTEXT #EVENT-GOAL<[MV-COMM-4779]>
:BG (:LF-EXISTS ?v6247*T-ORANGES  [X240] NIL
     (:AT ?v6247*T-ORANGES  [CORNING] [E242])))
```

This is similar to the previous call, only this time there exists a boxcar (`[X244]`, with empty restriction in the `LF-EXISTS` quantifier) which is being moved to Corning by the occurrence of an event of type `T-MOVE`. The input is again flagged as a goal due to the "need" modality in the utterance. The context is the node returned by the previous call, and is included because the cue phrase "so" implies a continuity between the previous utterance and this one, according to the dialogue manager. The `:BG` argument indicates background information arising from the relative clause "where there are oranges" that can be used by the plan reasoner to disambiguate possibilities.

Since the only event in the plan is of type `T-MOVE-COMMODITY`, and that cannot unify with the given event of moving a boxcar, the incorporation procedure starts searching the space of acceptable plans breadth-first. The sequence is shown graphically in Figure 4.11. Event nodes are represented by square boxes that also indicate role bindings. Fact nodes are represented by round boxes. The diagram (and subsequent plan graphs) omits temporal parameters although the plan reasoner ensures that they are added as needed to enforce the causal connections implied by the links.

The `T-Move-Car` event highlighted in the diagram unifies with the given event lambda expression, subject to certain assumptions. The result of incorporation is therefore the following set of plan description predicates, describing the new components of the plan shown in Figure 4.12. The context for future calls to `incorporate` is the event node labelled by `[MV-CAR-7867]` since this is the event in the plan corresponding to the lambda expression we were asked to incorporate.

```
((:PLAN-EVENT [F-MOVE-CAR MV-COMM-4779] [PLAN-4719])
 (:PLAN-GENERATES [F-MOVE-CAR MV-COMM-4779] [MV-COMM-4779]
  [PLAN-4719])
 (:PLAN-FACT
  (:AT [X244] [CORNING] [F-PRE1 [F-MOVE-CAR MV-COMM-4779]])
  [PLAN-4719])
 (:PLAN-ENABLES
  (:AT [X244] [CORNING] [F-PRE1 [F-MOVE-CAR MV-COMM-4779]])
  [F-MOVE-CAR MV-COMM-4779] [PLAN-4719])
 (:PLAN-EVENT [MV-CAR-7867] [PLAN-4719])
 (:PLAN-ACHIEVES [MV-CAR-7867]
  (:AT [X244] [CORNING] [F-PRE1 [F-MOVE-CAR MV-COMM-4779]])
  [PLAN-4719])
 (:PLAN-SUPPORTS (:APPLY
                  (:LAMBDA ?E*T-MOVE
                    (:AND (:ROLE ?E :R-AGENT [SYSHUM])
                          (:ROLE ?E :R-OBJECT [X244])
                          (:ROLE ?E :R-DST [CORNING])))
                  [MV-CAR-7867])
  [MV-CAR-7867] [PLAN-4719])
```

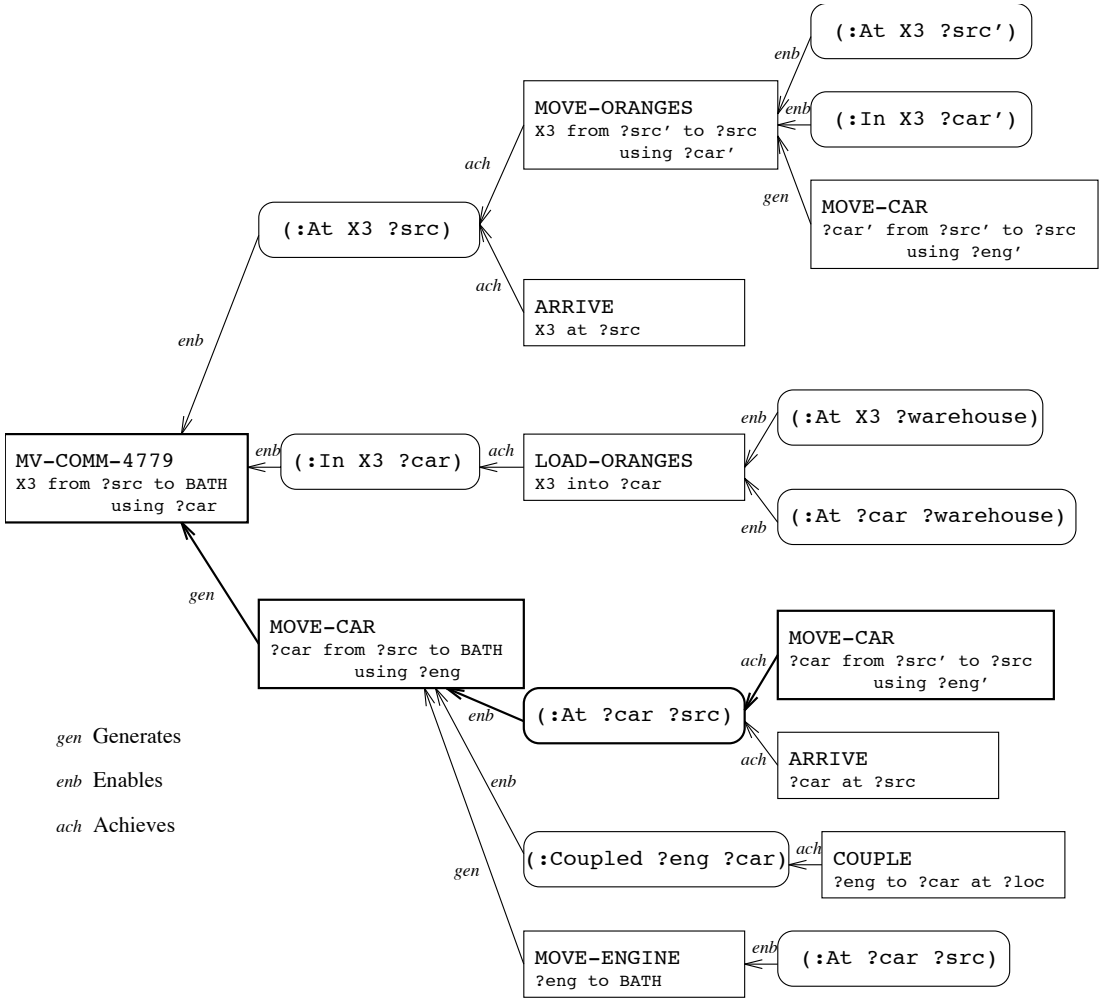


Figure 4.11: Expansion of plan graph during incorporatation

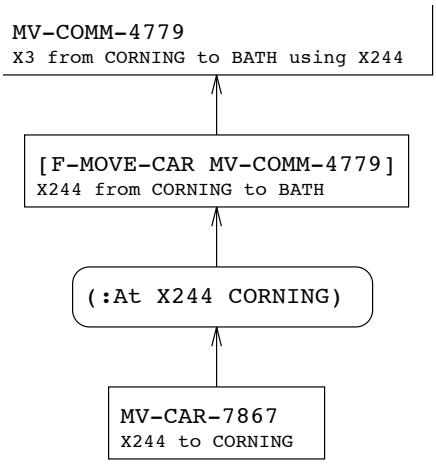


Figure 4.12: Plan graph after incorporating moving a boxcar to Corning

```
(:PLAN-FACT (:APPLY ...) [PLAN-4719])
(:PLAN-PREMISE (:APPLY ...) [PLAN-4719])
```

The manager then goes on to ask a question about the location of the oranges:

M: There are oranges at Corning. Right? (4)

Since the system already knows this (and in fact believes it is shared knowledge), it is also taken as an indirect suggestion to use the oranges at Corning in the plan. This results in the following call to the plan reasoner, where [ORANGES-1] is "the oranges at Corning," as determined by the scope and disambiguation module:

```
(INCORPORATE (:USE-OBJECT [ORANGES-1])
  [PLAN-4719]
  :INCORP-TYPE :ROLE-FILLER
  :FOCUS [ORANGES-1]
  :BG (:AT [ORANGES-1] [CORNING] [E529]))
```

This call is determined from the focus of the sentence, hence the attempt to incorporate use of the the oranges and not, for example, Corning. This focus information is indicated by the :FOCUS argument, although it is redundant in this case. Again, determination of focus is a function of the language and dialogue modules, not the plan reasoner.

The plan reasoner determines that the T-MOVE-COMMODITY event already in the plan (from the first utterance) has a role that can consistently be filled by the oranges. The following plan description predicates are returned indicating the additions to the plan:

```
((:PLAN-FACT (:AND (:EQ [ORANGES-1] [F-COMM MV-COMM-4779]))
  [PLAN-4719])
(:PLAN-SUPPORTS (:AND (:EQ [ORANGES-1] [F-COMM MV-COMM-4779]))
  [MV-COMM-4779] [PLAN-4719])
 (:PLAN-PREMISE (:AND (:EQ [ORANGES-1] [F-COMM MV-COMM-4779]))
  [PLAN-4719]))
```

This is a somewhat verbose way of stating that a new leaf node has been added to the plan supporting the MV-COMM-4779 node, which is itself returned as the new context node by incorporate.

Since the manager has released the turn by asking a question, the system meets its obligations by answering the question affirmatively and, in so doing, accepts the plan thus far and updates its belief spaces.

S: Right. (5)

The manager then suggests a course of action using a purpose construction:

M: So we need an engine to move the boxcar. Right? (6)

This results in the following call to the plan reasoner:

```
(INCORPORATE
  (:USE-OBJECT
    (:LAMBDA ?O*T-ENGINE (:DISC-MARKER [X710] ?O*T-ENGINE)))
```

```
[PLAN-4719]
:INCORP-TYPE :ROLE-FILLER
:CONTEXT #EVENT-GOAL<[MV-COMM-4779]>
:PURPOSE (:LF-EXISTS ?L*T-MOVE [E713]
          (:APPLY
           (:LAMBDA ?E*T-MOVE
            (:AND (:ROLE ?E :R-AGENT [SYSHUM])
                  (:ROLE ?E :R-OBJECT [X244])))
           ?L)
          (:OCCURS ?L))
:FOCUS (:LAMBDA ?O*T-ENGINE
        (:DISC-MARKER [X710] ?O*T-ENGINE)))
```

In this case, the object whose use we are being asked to incorporate is specified as a lambda expression, coming from the use of the indefinite article in "an engine." The `:DISC-MARKER` predicate is used by the dialogue manager and reference modules to track indefinites; as far the plan reasoner is concerned the engine being referred to is `[X710]`.

The processing of this utterance also illustrates the use of the `:PURPOSE` argument to `incorporate` (Section 4.5.2). The lambda expression describes an event of moving "the boxcar" identified previously (`[X244]`). The `incorporate` procedure first tries to match this `T-MOVE` event with the events already in the plan, and succeeds for `[MV-CAR-7867]`, the moving of a boxcar to Corning. It uses a simple heuristic (fewest assumptions) to prefer this over the other `T-MOVE-CAR` event, although it makes no difference in this case. The returned plan description predicates are simply:

```
((:PLAN-FACT (:DISC-MARKER [X710] [F-ENGINE MV-CAR-7867])
  [PLAN-4719])
 (:PLAN-SUPPORTS (:DISC-MARKER [X710] [F-ENGINE MV-CAR-7867])
  [MV-CAR-7867] [PLAN-4719])
 (:PLAN-PREMISE (:DISC-MARKER [X710] [F-ENGINE MV-CAR-7867])
  [PLAN-4719])
```

The resulting plan graph is still the same as that shown in Figure 4.12, except for the new role binding. The new context node returned by `incorporate` is the appropriate event node `#EVENT<[MV-CAR-7867]>`.

Since the incorporation succeeds, the system responds affirmatively and accepts the plan thus far, updating it's belief spaces:

`S: Right.` (7)

The manager then continues by identifying a particular engine:

`M: So there's an engine at Avon. Right?` (8)

In general, this could simply be a case of the manager informing the system about the state of the world. However, by querying its knowledge base, the system knows that the location of the engine is shared knowledge (it comes from the shared map). In that case, the utterance is taken to suggest use of the engine in the plan, resulting in the following call to the plan reasoner:

```
(INCORPORATE (:USE-OBJECT [X989])
  [PLAN-4719]
  :INCORP-TYPE :ROLE-FILLER
  :CONTEXT #EVENT<[MV-CAR-7867]>
  :FOCUS [X989]
  :BG (:AT [X989] [AVON] [E991]))
```

That is, incorporate use of the "the engine" (the discourse marker `[X989]`) in the plan rooted at the previously-returned context node. The background information is that the engine is useful because it is at Avon.

Again this is a trivial case of finding a role for the engine in one of the events in the plan. Since this is possible for `MV-CAR-7867` immediately, nothing else needs to be added to the plan beyond the role binding. The following plan description predicates indicate this:

```
((:PLAN-FACT (:EQ [X989] [F-ENGINE MV-CAR-7867]) [PLAN-4719])
 (:PLAN-SUPPORTS (:EQ [X989] [F-ENGINE MV-CAR-7867])
  [MV-CAR-7867] [PLAN-4719])
 (:PLAN-PREMISE (:EQ [X989] [F-ENGINE MV-CAR-7867])
  [PLAN-4719])
```

Again, the resulting plan graph is like Figure 4.12 with an additional role binding for the engine (which amounts to equating several discourse markers, as desired).

The system responds affirmatively, thereby both answering the literal question asked by the manager and accepting the suggestion:

```
S:  Right.                                                          (9)
```

The manager's next utterance is:

```
M:  So we should move the engine at Avon, engine E1, to Dansville (10)
    to pick up the boxcar there.
```

This results in the following call to `incorporate`, which indicates that moving the engine (`[ENGINE-1]`) is for the purpose of coupling with `[CAR-1]` ("the boxcar there," found by the reference module's querying the world knowledge after resolving "there" on linguistic grounds):

```
(INCORPORATE
   (:LF-EXISTS ?L*T-MOVE [E1193]
    (:APPLY
     (:LAMBDA ?E*T-MOVE
      (:AND (:ROLE ?E :R-AGENT [SYSHUM])
            (:ROLE ?E :R-OBJECT [ENGINE-1])
            (:ROLE ?E :R-DST [DANSVILLE])))
     ?L)
    (:OCCURS ?L))
  [PLAN-4719]
  :INCORP-TYPE :EVENT
```

```
:CONTEXT #EVENT<[MV-CAR-7867]>
:PURPOSE (:LF-EXISTS ?L*T-COUPLE [E1199]
          (:APPLY
           (:LAMBDA ?E*T-COUPLE
            (:AND (:ROLE ?E :R-AGENT [SYSHUM])
                  (:ROLE ?E :R-CAR [CAR-1])))
           ?L)
          (:OCCURS ?L)))
```

The `incorporate` procedure first attempts to incorporate the `T-COUPLE` event in the subgraph of the plan rooted at the `MV-CAR-7867` node. Although it doesn't match directly, a connection is found via the precondition for `T-MOVE-CAR` events that the car and engine involved be coupled. The context node returned from this purpose incorporation is the newly-created event `[COUPLE-16190]`, and the system then tries to incorporate moving the engine into this subplan. Since a precondition for coupling is that the engine and car be at the same location, the new event `[MV-ENGINE-16503]` is added to achieve that condition. Among other contraints propagated through the plan as a result of processing this utterance is the fact that the `[MV-CAR-7867]` event starts at Dansville, since that's where the coupling is taking place. The identity of the car and the engine are also extensively propagated through equality assertions added either as part of the event definitions or explicitly by the plan reasoner. The resulting plan graph is shown in Figure 4.13; `incorporate` returns plan description predicates corresponding to the new parts of the plan.

The system acknowledges the suggestion:

`S: Okay.` (11)

The manager then continues to refine her suggestion:

`M: And move it from Dansville to Corning.` (12)

This utterance seems somewhat unnatural because it is an edited version of the original dialogue in which the manager made a long, incremental suggestion. For the sample dialogue, we broke the utterance into separate contributions. Real incremental processing of user input including speech is one of the current research goals of the TRAINS project (see (Allen et al., 1995) for some more details). In any case, this supplementary utterance is treated as a separate suggestion, except that the initial "and" results in the dialogue manager passing the previous context node as the `:TEMP-SEQ` parameter to `incorporate`:

```
(INCORPORATE
   (:LF-EXISTS ?L*T-MOVE [E1649]
    (:APPLY
     (:LAMBDA ?E*T-MOVE
      (:AND (:ROLE ?E :R-AGENT [SYS])
            (:ROLE ?E :R-OBJECT [X1201])
            (:ROLE ?E :R-DST [CORNING])
            (:ROLE ?E :R-SRC [DANSVILLE])))
     ?L)
```

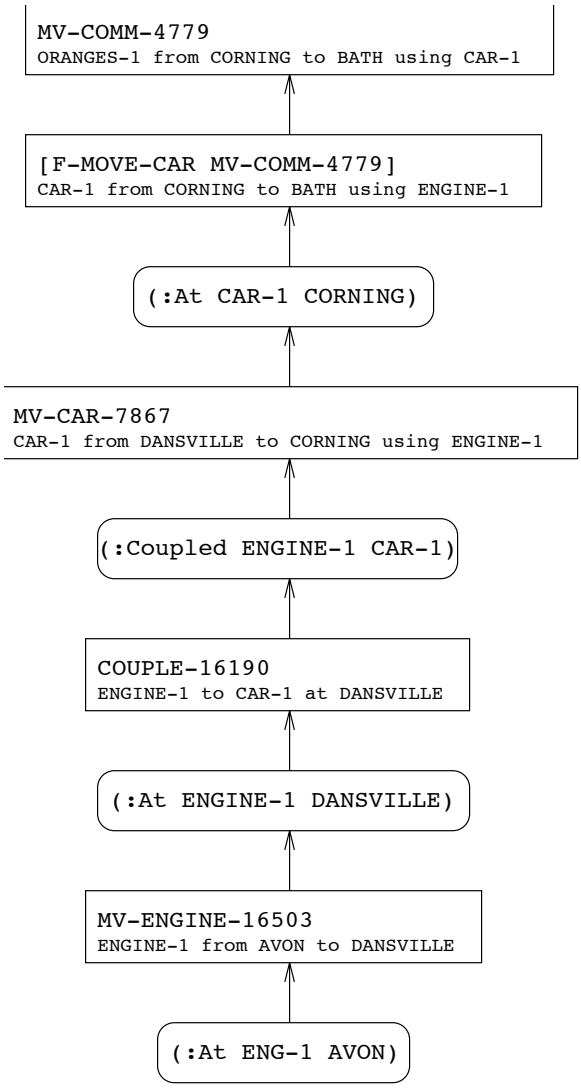


Figure 4.13: Plan graph after incorporating sending the engine to Dansville to pick up the boxcar

```
   (:OCCURS ?L))
  [PLAN-4719]
 :INCORP-TYPE :EVENT
 :TEMP-SEQ #EVENT<[MOVE-ENGINE-16503]>)
```

The idea is that the "and" used in this way indicates temporal sequencing of the events being described. The domain reasoner could then use this additional information to constrain the plan, although it is in fact unnecessary here since the events are already constrained by their causal relationship (or will be once the incorporation is done).

The incorporation in this case is trivial, since `[MV-CAR-7867]` is already known to be from Dansville to Corning and the rest of the constraints are consistent. The result is a set of plan description predicates for the new fact node representing the constraints. There are both a `PLAN-FACT` and a `PLAN-PREMISE` node in addition to the following

```
(:PLAN-SUPPORTS
  (:APPLY
   (:LAMBDA ?E*T-MOVE
    (:AND (:ROLE ?E :R-AGENT [SYS])
          (:ROLE ?E :R-OBJECT [X1201])
          (:ROLE ?E :R-DST [CORNING])
          (:ROLE ?E :R-SRC [DANSVILLE])))
   [MV-CAR-7867])
  [MV-CAR-7867] [PLAN-4719])
```

The resulting plan graph is a minor modification of Figure 4.13.

The manager continues without pausing:

```
M:  Load up some oranges into the boxcar.                    (13)
```

This results in the following call:

```
(INCORPORATE
   (:LF-EXISTS ?v19300*T-ORANGES  [X1785] NIL
    (:LF-EXISTS ?L*T-LOAD  [E1791]
     (:APPLY
      (:LAMBDA ?E*T-LOAD
       (:AND (:ROLE ?E :R-AGENT [SYS])
             (:ROLE ?E :R-COMMODITY ?v19300)
             (:ROLE ?E :R-CONTAINER [X714])))
      ?L)
     (:OCCURS ?L)))
   [PLAN-4719]
  :INCORP-TYPE :EVENT)
```

The result of the call is that a new event, `[LD-ORANGES-21221]`, is added to the plan to achieve one of the enablers of moving the oranges (`[MV-COMM-4779]`), namely that the oranges be in the boxcar. The resulting plan is shown in Figure 4.14 and `incorporate` returns appropriate plan description predicates to describe the additions to the plan.

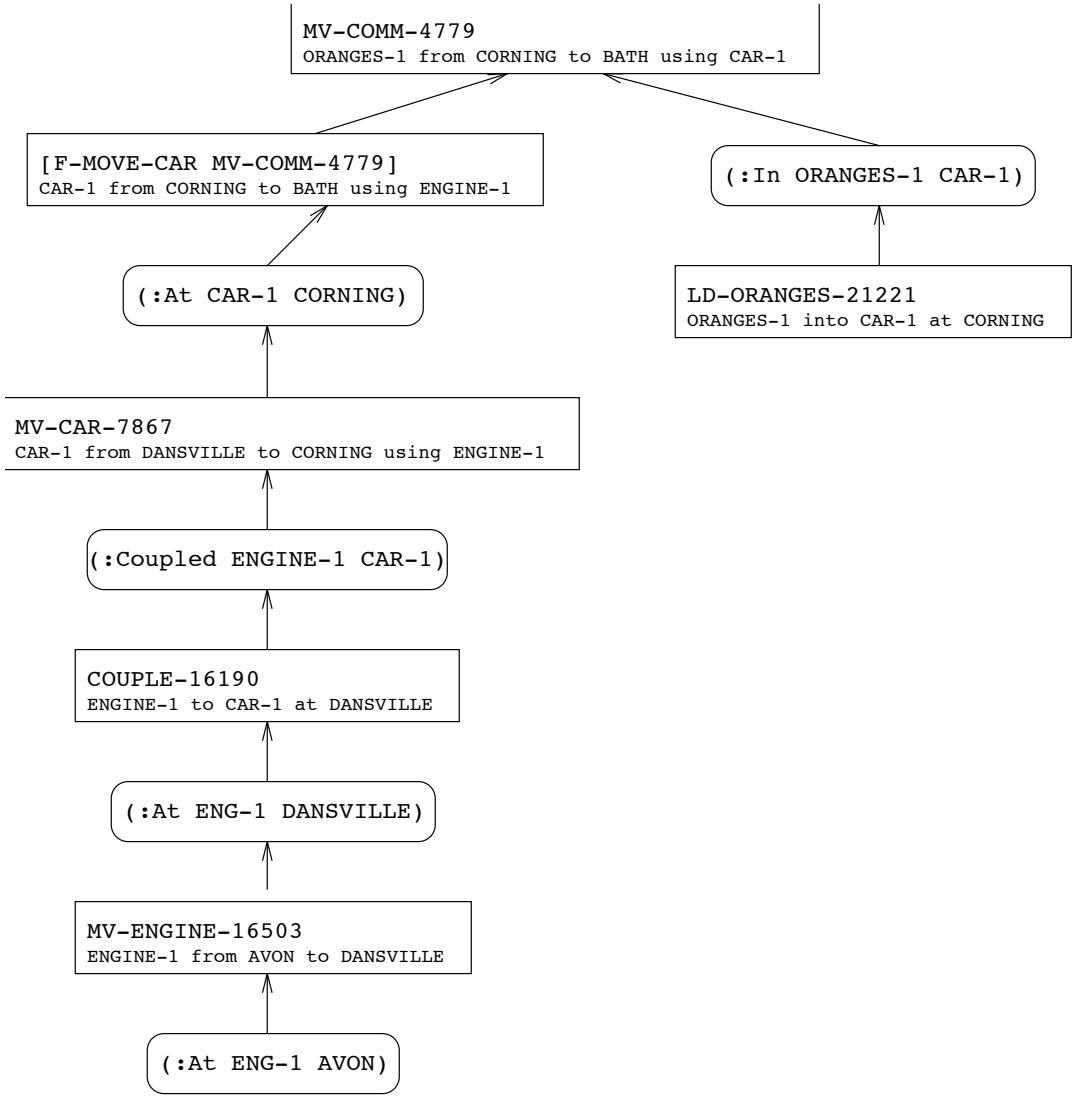


Figure 4.14: Plan graph after incorporating loading the oranges

Finally, the manager concludes the multi-utterance suggestion with:

```
M:  And then move it on to Bath.                                    (14)
```

This results in another call to `incorporate` using the `:TEMP-SEQ` parameter, this time the moving to Bath is sequenced after the loading of the oranges:

```
(INCORPORATE
   (:LF-EXISTS ?L*T-MOVE  [E1909]
    (:APPLY
     (:LAMBDA ?E*T-MOVE
      (:AND (:ROLE ?E :R-AGENT [SYS])
            (:ROLE ?E :R-OBJECT [X1201])
            (:ROLE ?E :R-DST [BATH])))
     ?L)
    (:OCCURS ?L))
   [PLAN-4719]
  :INCORP-TYPE :EVENT
  :TEMP-SEQ #EVENT<[LD-ORANGES-21221]>)
```

This moving event unifies uniquely with the `T-MOVE-CAR` event that generates `[MV-COMM-7867]`, so the only thing that needs to be added to the plan are the unifying assumptions:

```
(:PLAN-SUPPORTS
  (:APPLY
   (:LAMBDA ?E*T-MOVE
    (:AND (:ROLE ?E :R-AGENT [SYS])
          (:ROLE ?E :R-OBJECT [X1201])
          (:ROLE ?E :R-DST [BATH])))
   [F-MOVE-CAR MV-COMM-4779])
 [F-MOVE-CAR MV-COMM-4779] [PLAN-4719])
```

The `PLAN-FACT` and `PLAN-PREMISE` predicates are also returned, and the resulting plan graph is trivial addition to that shown in Figure 4.14.

The system acknowledges all three suggestions when it gets the turn, since all three incorporations succeeded:

```
S:  Okay.                                                           (15)
```

The manager then asks a linguistically tricky question:

```
M:  How does that sound?                                            (16)
```

Presumably this is a meta-query where "that" refers to either the previous suggestion(s) or the plan as a whole. The language modules determine the meaning of the pronoun, and the dialogue manager determines that it should call the domain plan reasoner to elaborate the plan. In this case, the plan is almost fully-specified already, and in fact this is typical of our dialogues.

As described in Section 4.5.3, the system decomposes events that do not correspond directly to actions, in this case adding `T-MOVE-ENGINE` events to generate each `T-MOVE-CAR` event. It also checks that all enablers of the events are added to the plan,

for example using the fact that there is an car (`CAR-1`) at Dansville initially to enable both the coupling with the engine (`ENGINE-1`) and the moving the boxcar from Dansville to Corning. It also verifies and adds preconditions achieved by the plan, for example, the fact that the car is subsequently at Corning enables both moving it to Bath and loading it at Corning. The former link was already in the plan, the latter is addded during elaboration. The resulting plan graph is shown in Figure 4.15. Event nodes corresponding to actions (that will be used by the execution planner) are shaded.

Since in this case there are no ambiguities reported, the system responds with:

```
S:  That's no problem.                                          (17)
```

The manager responds:

```
M:  Good.                                                       (18)
```

thereby accepting the completed plan. The system dispatches it for execution by the execution planner in the simulated TRAINS world and concludes the dialogue with the manager:

```
S:  The plan has been fully specified and executed.  Goodbye.   (19)
```

4.7 Discussion and Future Work

This section describes some of the outstanding issues for the TRAINS domain plan reasoner and the TRAINS system.

As noted in Section 2.6 regarding the event-based temporal logic, the inability to represent and reason with metric time is a serious shortcoming, particularly in the TRAINS domain. It is hoped that by using a combined interval-metric temporal reasoner (Gerevini and Schubert, 1994; Gerevini et al., 1994) we can express the required metric constraints. Issues remain, such as use of the *Disjoint* relation so important for planning or reasoning about doing something "long enough." Other forms of constraint-based reasoning, such as capacities and quantities also need to be better integrated, and presumably reasoned about using optimized specialists and not first-principles axiomatizations.

Another shortcoming of the plan reasoner is that it effectively reasons without projecting the consequences of its actions. This leads to problems in effectively determining whether preconditions hold at required times, although the assumption-based unification covers some of the cases more traditionally handled by projection. Work is currently underway on the construction of a probabilistic projection module based on the interval temporal logic (Yampratoom, 1994). This would support plan projection as well as stochastic simulation of plan execution to gather statistics for future decision-making.

A shortcoming of the plan reasoner that illustrates the loose fit between theory and practice is the fact that the plan reasoner does not perform the kind of explanation closure reasoning that figures so prominently in both theories of representation. The assumption-based unification of temporal terms takes into account the consistency of domain constraints, but this only addresses part of the issue. Although this is clearly inadequate and a significant discrepancy between the theory and the implementation,

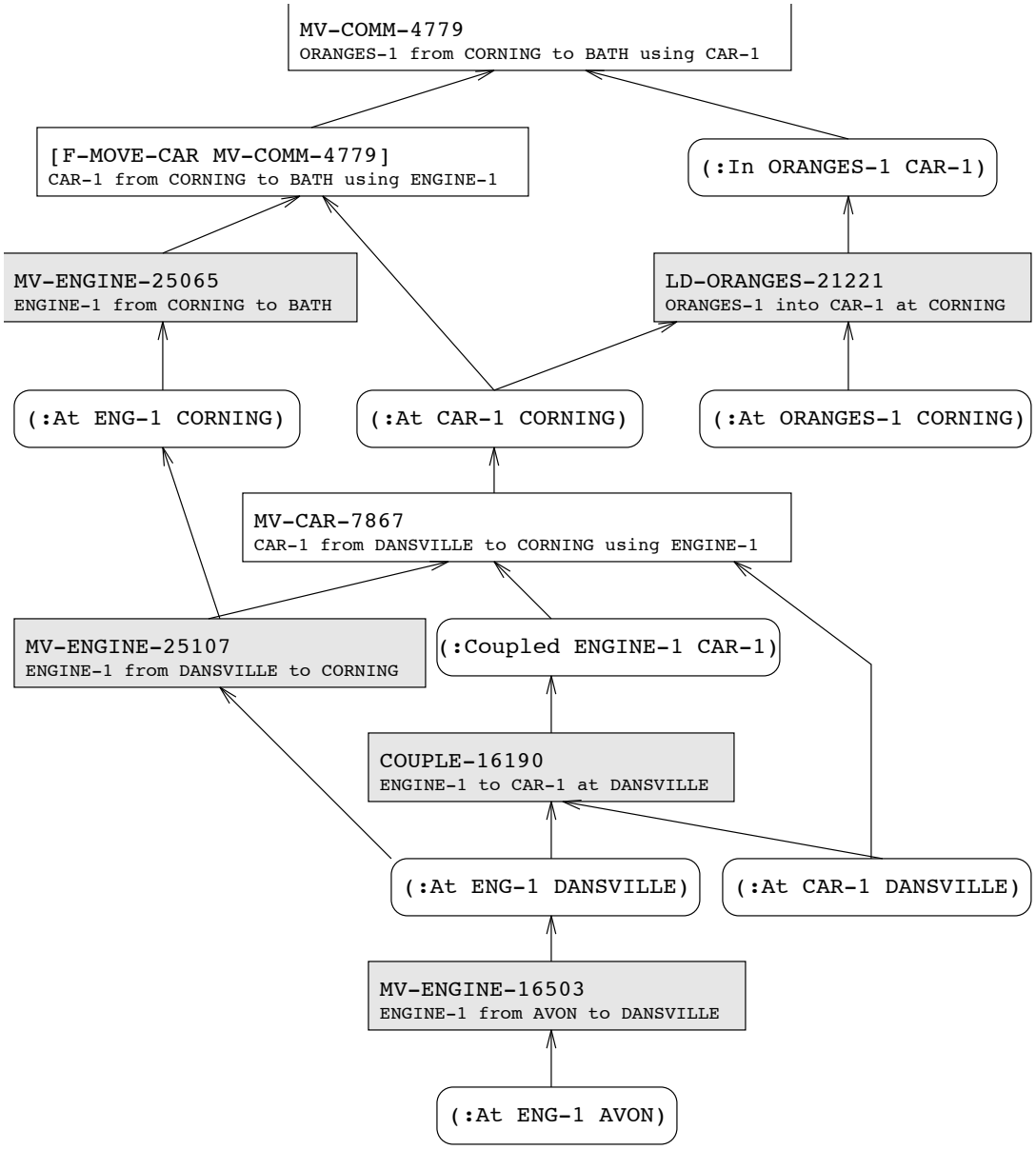


Figure 4.15: Final plan for TRAINS-93 sample dialogue

implementing such reasoning effectively was simply not possible given the tools we were working with. Some separate work on performing explanation closure reasoning using EBTL has been done, but it remains a major shortcoming of the current implementation.

One of the design goals of the TRAINS system as an intelligent planning assistant was that it would be able to manage multiple plans and other scheduled events, and coordinate activities requested by the manager with these other activities. The system as it is currently implemented can really only discuss a single plan, which it dispatches for execution once it is complete. Note however that the formalism of course supports external events (*cf.* Section 2.4), and the plan reasoner can make use of their occurrence in the plan. And the plan reasoner can also reason about different plans simply by using different terms to denote them, although it is not able to understand the possible relationships and interactions between these plans explicitly.

As noted in the discussion of elaboration (Section 4.5.3), the plan reasoner is currently unable to resolve most ambiguities, even fairly trivial ones, and hence is not very "intelligent." While it would be straightforward to hack in all kinds of Lisp mechanisms to resolve such questions, what is really needed is a principled way of specifying preferences between choices and reasoning with these. Some of this will no doubt be based on the preference mechanisms underlying argumentation (Section 3.2), some of it will come from statistical information from the executor or projector modules, and some of it will require new interfaces and meta-discussion abilities. As we commented in Section 3.6.3, such meta-reasoning and meta-communication is also an important area for future work on the formal theory of the plans-as-arguments representation.

Finally, following from that last point, the TRAINS system as a whole needs to be made much more robust and able to participate in conversations at many different levels. Since plan reasoning is already a form of meta-reasoning about facts and events and their relations in terms of arguments, this will require some kind of meta-meta-reasoning and the linguistic means or other forms of interaction for specifying them. The next generation of the TRAINS system has made robustness a primary goal, to the extent of sacrificing depth of theoretical purity somewhat in exchange for breadth of coverage and interactivity. The plan reasoner will need to evolve to provide the new services required by such a system.

4.8 Summary and Contributions

The TRAINS-93 domain plan reasoner was intended to fulfill two somewhat contradictory goals. First, it embodies the theories of knowledge representation for actions, events, and time and for plans decribed in Chapters 2 and 3 respectively. In this regard, it is an approximation of the full power of the theories, limited by both pragmatic factors and theoretical bounds. Second, it is part of a functional, albeit somewhat limited, interactive planning system. I feel that it makes a contribution in both areas.

As embodiment of the theory of knowledge representation based on interval temporal logic (Chapter 2), it provides good evidence regarding the suitability of the representation for building realistic reasoning systems. As well, it illustrates aspects of the

representation that are computationally problematic (although of course the logic is only semi-decidable in general). It seems to us that both of these are important aspects of building AI theories based on logic. We cannot stop at theoretical results like soundness and completeness. Rather, we have to get out there and build systems that *use* the logic, even if they don't actually *do* logic (that is, they aren't necessarily theorem provers). Sub-parts of the logic may be amenable to special treatment via specialized reasoners, and our implementation leads to their improvement and the development of new specialists.

This relationship between theory and practice is also illustrated by the relationship between the system's representation of plans and the formal representation of plans as arguments (Chapter 3). On the one hand, since the representation based on arguments was motivated by considering plan reasoning as a dialectical process, it fits gracefully into the design of an interactive system. On the other hand, although the plan description predicates would allow a pure logic programming approach to building such a system, pragmatic factors such as controlling search and using information from failed searches led us to design the plan reasoner as a heuristic search procedure informed by the logic of arguments instead.

Finally, of course, pragmatic factors were the main consideration in the development of the plan reasoner as a module of the overall system. Specification of interfaces suitable for use by other modules and languages for exchanging information between modules were paramount, and sometimes governed by factors not of direct relevance to the plan reasoner. The handling of purpose clauses by the plan reasoner (Section 4.5.2) is a case in point. Although the plan reasoner has its limitations, especially as regards elaboration (Section 4.5.3), it is nonetheless a working component in a sophisticated, if brittle, interactive plan reasoning system. It is quite general within its relatively narrow range of applicability, and should serve as a solid foundation for the next generation of the TRAINS system, already under development.

5 Conclusions

This chapter summarizes the contributions of the three aspects of this dissertation and of the dissertation as a whole.

5.1 Reasoning about Actions and Events

The event-based interval temporal logic provides an expressive yet natural representation of time, properties, events and actions that allows us to formalize realistic domains without rendering them uninteresting. The work described in Chapter 2 builds on the extensive previous work on using the logic to reason about action. In particular, it revisits the formal foundations of the representation and clarifies its use in causal reasoning. The adaptation of the explanation closure technique to the temporal logic as I have presented it is natural and yet captures important knowledge about the domain that we need for other purposes than just prediction. The fact that the assumptions underlying prediction are explicit in this approach is used to advantage in reasoning about plans.

The application of the temporal logic to the Sandewall test suite is important for comparing the temporal approach to other representations of time and action. While the problems hardly challenge our formalism, at least we can see where everyone stands. As well, considering these problems and other solutions allows us to compare the explanation closure approach with various nonmonotonic approaches to reasoning about action, a comparison that was again continued in our discussion of plan representations. The discussion of external events and simultaneous actions combined previous work on temporal logic approaches to these issues within a unified framework, and considered them in the context of the sorts of real-world problems we are interested in when building realistic planning systems.

5.2 Plan Representation and Reasoning for Mixed-Initiative Planning

The representation of plans as arguments described in Chapter 3 constitutes a reappraisal of what plans are and of their role in intelligent, interactive behaviour. Although

the presentation has left many questions unanswered, there have been some concrete contributions.

First I used data collected from real mixed-initiative planning scenarios to try and define exactly what is mixed-initiative planning. As the field attempts to define itself, my study shows that a broad perspective is necessary to do justice to the concept. The emphasis on communication and defeasibility in mixed-initiative planning led to my specification of a formal system of defeasible reasoning based on arguments that borrows from several sources and that I compared to alternative forms of nonmonotonic reasoning. I then applied argument-based reasoning to reasoning about plans by considering plans to be arguments that a certain course of action under certain conditions would achieve certain goals. In developing a treatment of the frame and qualification problems, I would like to that think I have combined both philosophical and AI planning perspectives. For concreteness, I presented a reconstruction of traditional STRIPS planning within the defeasible reasoning system which, I think, sheds some new light on both of them.

As I've stated repeatedly, the work described in Chapter 3 is only the beginning of the study of argumentation and plan reasoning. I hope that the foundations laid here together with the discussions of related work and outstanding issues are sufficient to at least validate the plausibility of the idea of plans-as-arguments and justify further work on it in the context of mixed-initiative planning.

5.3 The TRAINS-93 Domain Plan Reasoner

The TRAINS domain plan reasoner is a working component in a sophisticated, if brittle, interactive planning system. All the modules in the system, from the parser to the executor are based on theories designed not for easy implementation but for theoretical clarity and breadth of coverage. Although the implementations are always approximations of the theories, they are informed by and in turn inform the theoretical work. It is this combination of theory and practice that, to my mind, separates AI from philosophy, linguistics, or psychology.

The particular contributions of my work on the domain plan reasoner include the development and refinement of the underlying knowledge representation system (Rhet) and the specification and implementation of an abstract interface between this system and the plan reasoner (EBTL). The plan recognition component is flexible enough to support a wide range of phenomena encountered in mixed-initiative planning systems like TRAINS. It supports a variety of interfaces, reasons from arbitrary knowledge contexts, and returns useful information from both successful and failed attempts to incorporate something into a plan. Although the elaboration module is not nearly as capable, it is minimally competent and, in fact, in dialogues such as ours much of the reasoning burden is borne by the recognition component. As well, since we are working with a more expressive representation language, many of the choices encountered by the planner are not about trivial orderings but are rather real domain-dependent questions, in some sense.

In any case, since the system is not expected to be omniscient as previous AI systems have been expected to be, there will always be things that it does not know or cannot determine given its limited resources. Since the system is interactive, it can always ask a question or offer a choice rather than risk making a bad decision or wasting resources. Recognizing this, as the TRAINS domain plan reasoner (and the entire TRAINS system) does is an important aspect of building mixed-initiative planning systems.

5.4 Dissertation Conclusion

The thesis of this dissertation was essentially that we need sufficiently expressive representations of time, properties, events, actions, and plans in order to build systems that can participate in natural, realistic mixed-initiative planning situations. I believe that the three parts of the dissertation have both illustrated this need and proposed representations that address it. Taken together, they represent steps, some more tentative than others, towards an adequate knowledge representation for mixed-initiative planning.

Bibliography

Allen, James F. 1979. *A plan-based approach to speech act recognition.* Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Ont.

Allen, James F. 1983a. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843. Also in *Readings in Knowledge Representation*, R.J. Brachman and H.J. Levesque (eds.), Morgan Kaufmann, 1985, pp. 509–522.

Allen, James F. 1983b. Recognizing intentions from natural language utterances. In M. Brady and R. Berwick, editors, *Computational Models of Discourse*. MIT Press, Cambridge, MA.

Allen, James F. 1984. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 464–479.

Allen, James F. 1991a. Planning as temporal reasoning. In James Allen, Richard Fikes, and Eric Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 3–14. Morgan Kaufmann, Cambridge, MA.

Allen, James F. 1991b. Temporal reasoning and planning. In *Reasoning about Plans*, pages 1–68. Morgan Kaufmann, San Mateo, CA.

Allen, James F. and George Ferguson. 1994. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579. A much earlier version appeared in Working notes of the Second Symposium on Logical Formalizations of Commonsense Reasoning, Austin, TX, 11–13 January, 1993.

Allen, James F. and Patrick J. Hayes. 1989. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5(4):225–238.

Allen, James F. and Johannes A. Koomen. 1983. Planning using a temporal world model. In Alan Bundy, editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 741–747. William Kaufmann, Karlsruhe, West Germany. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 559–565.

Allen, James F. and C. Raymond Perrault. 1980. Analyzing intention in utterances. *Artificial Intelligence*, 15(3):143–178.

Allen, James F., Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. 1995. The TRAINS project: A case study in defining a conversational planning agent. *Journal of Experimental and Theoretical AI*, 7:7–48. Also available as TRAINS Technical Note 93-4, Department of Computer Science, University of Rochester.

Allen, J.F. and B.W. Miller. 1989. The rhetorical knowledge representation system: A user's manual (for rhet version 15.25). Technical Report 238 (revised), Department of Computer Science, University of Rochester, Rochester, NY.

Austin, John L. 1962. *How to Do Things With Words*. Harvard University Press, Cambridge, MA.

Bacchus, Fahiem, Josh Tenenberg, and Johannes A. Koomen. 1989. A non-reified temporal logic. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 2–10. Toronto, Ont.

Baker, Andrew. 1991. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–24.

Balkanski, Cecile T. 1990. Modelling act-type relations in collaborative activity. Technical Report 23-90, Center for Research in Computing Technology, Harvard University, Cambridge, MA.

Brown, Frank M., editor. 1987. *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*. Lawrence, KA, Morgan Kaufmann.

Carberry, Sandra. 1990. *Plan Recognition in Natural Language*. MIT Press, Cambridge, MA.

Carberry, Sandra. 1991. Incorporating default inferences into plan recognition. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 471–478. Sydney, Australia.

Cavalli-Sforza, Violetta and Daniel D. Suthers. 1994. Belvedere: An environment for practicing scientific argumentations. In Ronald P. Loui and Thomas Gordon, editors, *Working Notes of the AAAI-94 Workshop on Computational Dialectics*. Seattle, WA.

Chapman, David. 1987. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 537–558.

Cheeseman, Peter. 1988. An inquiry into computer understanding. *Computational Intelligence*, 4(1):58–66.

Chu-Carroll, Jennifer and Sandra Carberry. 1994. A plan-based model for response generation in collaborative plan-based dialogues. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 799–805. AAAI Press, Seattle, WA.

Cohen, Philip R. 1978. On knowing what to say: Planning speech acts. Technical Report 118, Department of Computer Science, University of Toronto, Toronto, Ont.

Cohen, Philip R. and C. Raymond Perrault. 1979. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212. Also in *Readings in Artifical Intelligence*, B. L. Webber and N. J. Nilsson (eds.), 1981, pp. 478–495.

Darwiche, Adnan and Judea Pearl. 1994a. Symbolic causal networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 238–244. AAAI Press, Seattle, WA.

Darwiche, Adnan and Judea Pearl. 1994b. Symbolic causal networks for reasoning about actions and plans. In *Working Notes of the AAAI Spring Symposium on Decision-Theoretic Planning*, pages 41–47. Stanford, Ca.

Davidson, Donald. 1967. The logical form of action sentences. In N. Rescher, editor, *The Logic of Decision and Action*. University of Pittsburgh Press. Excerpted in *The Logic of Grammar*, D. Davison and G. Harmon (eds.), Dickenson Publishing Co., 1975, pp. 235–245.

Davis, Ernest. 1992. Infinite loops in finite time: Some observations. In Bernard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 47–58. Morgan Kaufmann, Boston, MA.

Dean, Tom and Drew McDermott. 1987. Temporal data base management. *Artificial Intelligence*, 32:1–55. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 596–623.

Dowty, David R. 1986. The effects of aspectual class on the temporal structure of discourse: Semantics or pragmatics? *Linguistics and Philosophy*, 9(1).

Doyle, John. 1980a. *A Model for Deliberation, Action, and Introspection*. Ph.D. thesis, Massachusetts Institute of Technology, Cambidge, MA.

Doyle, John. 1980b. A truth maintenance system. *Artificial Intelligence*, 12:231–272. Also in *Readings in Nonmonotonic Reasoning*, M.L. Ginsberg (ed.), Morgan Kaufmann, 1987, pp. 259–279.

Elkan, C. 1990. Incremental, approximate planning. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 145–150. Boston, MA.

Eshghi, Kave. 1988. Abductive planning with event calculus. In R.A. Kowalski and K.A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 562–579. MIT Press, Cambridge, MA.

Ferguson, George. 1992. Explicit representation of events, actions, and plans for assumption-based plan reasoning. Technical Report 428, Department of Computer Science, University of Rochester, Rochester, NY.

Ferguson, George and James F. Allen. 1993. Cooperative plan reasoning for dialogue systems. In *Papers from the AAAI-93 Fall Symposium on Human-Computer Collaboration: Reconciling Theory, Synthesizing Practice, AAAI Technical Report FS-93-05*. Raleigh, NC.

Ferguson, George and James F. Allen. 1994. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 43–48. Chicago, IL.

Fikes, Richard E. and N.J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:198–208. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 88–97.

Gelfond, Michael, Vladimir Lifschitz, and Arkady Rabinov. 1991. What are the limitations of the situation calculus? In *Proceedings of the AAAI Symposium on Logical Formalizations of Commonsense Reasoning*, pages 59–59. Stanford University.

Georgeff, Michael P. 1986a. Actions, processes, and causality. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, Los Altos, CA.

Georgeff, Michael P. 1986b. The representation of events in multiagent domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 70–75. University of Pennsylvania, Philadelphia, PA.

Gerevini, Alfonso and Lenhart Schubert. 1994. Efficient algorithms for qualitative reasoning about time. Technical Report 496, Computer Science Department, University of Rochester, Rochester, NY.

Gerevini, Alfonso, Lenhart Schubert, and Stephanie Schaeffer. 1994. The temporal reasoning systems timegraph i-ii. Technical Report 494, Computer Science Department, University of Rochester, Rochester, NY.

Ginsberg, Matthew L. 1994a. Approximate planning. *Artificial Intelligence*. To appear.

Ginsberg, Matthew L. 1994b. Approximate planning (extended abstract). In Kristian Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*. University of Chicago, Chicago, IL.

Goebel, Randy G. and Scott D. Goodwin. 1987. Applying theory formation to the planning problem. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 207–232. Lawrence, KA, Morgan Kaufmann.

Goldman, Alvin I. 1970. *A Theory of Human Action*. Prentice-Hall, Englewood Cliffs, NJ.

Goodwin, S.D. 1991. *Statistically Motivated Defaults*. Ph.D. thesis, Department of Computer Science, University of Alberta, Edmonton, Alta. Also available as U. of Regina TR CS-91-03.

Gordon, Thomas F. 1993. *The Pleadings Game: An Artificial Intelligence Model of Procedural Justice*. Ph.D. thesis, TU Darmstadt, Darmstadt, Germany.

Green, Cordell. 1969. An application of theorem proving to problem solving. In Donald E. Walker, editor, *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 741–747. Washington, DC. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 67–87.

Grice, H. Paul. 1975. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics, vol. 3, Speech Acts*, pages 41–58. Academic Press, New York.

Gross, Derek, James F. Allen, and David R. Traum. 1993. The TRAINS-91 dialogues. TRAINS Technical Note 92-1, Department of Computer Science, University of Rochester, Rochester, NY, 14627.

Grosz, Barbara J. and Candice L. Sidner. 1990. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press.

Haas, Andrew R. 1992. A reactive planner that uses explanation closure. In Bernard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 93–102. Morgan Kaufmann, Boston, MA.

Haas, Andrew W. 1987. The case for domain-specific frame axioms. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 343–348. Lawrence, KA, Morgan Kaufmann.

Hamblin, C. L. 1972. Instants and intervals. In J. T. Fraser, F. C. Haber, and G. H. Müller, editors, *The Study of Time*, pages 324–328. Springer-Verlag, New York.

Hanks, S. and D. McDermott. 1986. Default reasoning, nonmonotonic logic, and the frame problem. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 328–333. University of Pennsylvania, Philadelphia, PA.

Hwang, Chung Hee and Lenhart K. Schubert. 1993a. Episodic logic: A comprehensive, natural representation for language understanding. *Minds and Machines*, 3:381–419.

Hwang, Chung Hee and Lenhart K. Schubert. 1993b. Episodic logic: A situational logic for natural language processing. In P. Aczel, D. Israel, Y. Katagiri, and S. Peters, editors, *Situation Theory and its Applications*, volume 3. CSLI, Stanford, CA.

Jearl, Judea and T.S. Verma. 1991. A theory of inferred causation. In James Allen, Richard Fikes, and Eric Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 441–453. Morgan Kaufmann, Cambridge, MA.

Kambhampati, Subbarao. 1989. *Flexible reuse and modification in hierarchical planning: A validation structure based approach.* Ph.D. thesis, Department of Computer Science, University of Maryland, College Park, MD. Available as CS Technical Report 2334, Dept. of Computer Science, University of Maryland.

Kambhampati, Subbarao. 1994. Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10(2):212–245.

Kambhampati, Subbarao and James A. Hendler. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258.

Kautz, Henry A. 1986. The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 401–405. University of Pennsylvania, Philadelphia, PA.

Kautz, Henry A. 1987. *A Formal Theory of Plan Recognition.* Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. Available as Technical Report 215.

Kautz, Henry A. 1991. A formal theory of plan recognition and its implementation. In *Reasoning about Plans*, pages 69–126. Morgan Kaufmann, San Mateo, CA.

Keynes, John Maynard. 1921. *A Treatise on Probability.* Macmillan and Co., London.

Knoblock, Craig A. 1992. An analysis of ABSTRIPS. In *Proceedings of the First International Conference on AI Planning Systems*, pages 126–135. Morgan Kaufmann, College Park, MD.

Konolige, Kurt. 1988. Defeasible argumentation in reasoning about events. In Zbigniew W. Ras and Lorenza Saitta, editors, *Methodologies for Intelligent Systems 3, Proceedings of the Third International Symposium*, pages 380–390. North-Holland, Turin, Italy.

Konolige, Kurt and Martha E. Pollack. 1989. Ascribing plans to agents, preliminary report. In Natesa Sastri Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 924–930. Detroit, MI.

Koomen, Johannes A.G.M. 1989. The TIMELOGIC temporal reasoning system in Common Lisp. Technical Report 231, Computer Science Department, University of Rochester, Rochester, NY. Revised.

Kowalski, Robert. 1992. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146.

Kowalski, Robert and Marek Sergot. 1986. A logic-based calculus of events. *New Generation Computing*, 4:67–95.

Kushmerick, Nicholas, Steve Hanks, and Daniel Weld. 1994. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press, Seattle, WA.

Kyburg, Henry E., Jr. 1974. *The Logical Foundations of Statistical Inference*. D. Reidel, Dordrecht.

Kyburg, Henry E., Jr. 1990. Probabilistic inference and probabilistic reasoning. *Philosophical Topics*, 18(2):107–116.

Kyburg, Henry E., Jr. 1994. Believing on the basis of the evidence. *Computational Intelligence*, 10(1):3–20.

Kyburg Jr., Henry E. 1991. Beyond specificity. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *Uncertainty in Knowledge Bases*, pages 204–212. Springer Verlag.

Ladkin, Peter and R. Maddux. 1988. Representation and reasoning with convex time intervals. Technical report KES.U.88.2, Kestrel Institution, Palo Alto, CA.

Lam, Wai and Fahiem Bacchus. 1994. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(3):270–293.

Lambert, Lynn and Sandra Carberry. 1991. A tripartite plan-based model of dialogue. In *Proceedings of the Twenty-Ninth Annual Meeting of the Association for Computational Linguistics (ACL-91)*, pages 47–54. University of California, Berkeley, CA.

Lifschitz, Vladimir. 1987. Formal theories of action. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 35–58. Lawrence, KA, Morgan Kaufmann.

Lifschitz, Vladimir and Arkady Rabinov. 1989. Miracles in formal theories of action (research note). *Artificial Intelligence*, 38(2):225–238.

Lin, F. and Y. Shoham. 1989. Argument systems: A uniform basis for nonmonotonic reasoning. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 245–255. Morgan Kaufmann, Toronto, Ont.

Lin, Fangzhen and Yoav Shoham. 1992. Concurrent actions in the situation calculus. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 580–585. San Jose, CA.

Litman, Diane. 1985. *Plan recognition and discourse analysis: an integrated approach for understanding dialogues*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. Available as Technical Report 170.

Litman, Diane and James F. Allen. 1987. A plan-recognition model for subdialogues in conversations. *Cognitive Science*, 11(2).

Lochbaum, Karen E. 1991. An algorithm for plan recognition in collaborative discourse. In *Proceedings of the Twenty-Ninth Annual Meeting of the Association for Computational Linguistics (ACL-91)*, pages 33–38. University of California, Berkeley, CA.

Lochbaum, Karen E., Barbara J. Grosz, and Candace L. Sidner. 1990. Models of plans to support communication: An initial report. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 485–490. Boston, MA.

Loui, Ronald. 1990a. Defeasible specification of utilities. In Henry E. Kyburg, Jr., Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 345–360. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Loui, Ronald P. 1990b. Defeasible decisions: What the proposal is and isn't. In M. Henrion, R. D. Schacter, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 99–116. Elsevier Science Publishers (North-Holland).

Loui, Ronald P. 1991a. Argument and belief: Where we stand in the keynesian tradition. *Minds and Machines*, 1:357–365.

Loui, Ronald P. 1991b. Dialectic, computation, and ampliative inference. In R. Cummins and J. Pollock, editors, *Philosophy and AI*. MIT Press, Cambridge, MA.

Loui, R.P. 1987. Defeat among arguments: A system of defeasible inference. *Computational Intelligence*, 3(2):100–106.

Martin, Nathaniel G. 1993. *Using Statistical Inference to Plan Under Uncertainty*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. To appear as a Technical Report.

Martin, Nathaniel G. and James F. Allen. 1993. Statistical probabilities for planning. Technical Report 474, Department of Computer Science, University of Rochester, Rochester, NY.

McAllester, David and David Rosenblitt. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639. MIT Press.

McCarthy, J. 1980. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39.

McCarthy, John and Patrick J. Hayes. 1969. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. American Elsevier Publishing Co., Inc. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 393–435.

McDermott, Drew. 1985. Reasoning about plans. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–318. Ablex Publishing, Norwood, NJ.

Miller, Rob and Murray Shanahan. 1994. Narratives in the situation calculus. *Journal of Logic and Computation, Special Issue on Actions and Processes.*

Moore, Robert C. 1985. A formal theory of knowledge of action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World.* Ablex Publishing, Norwood, NJ. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 480–520.

Morgenstern, Leora and Lynn A. Stein. 1988. Why things go wrong: A formal theory of causal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88).* University of Minnesota, St. Paul, MN. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 641–646.

Mourelatos, A.P.D. 1978. Events, processes and states. *Linguistics and Philosophy*, 2:415–434.

Nebel, Bernhard and Christer Bäckström. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence*, 66:125–160.

Nilsson, Nils J. 1980. *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kauffman, San Mateo, CA.

Pednault, Edwin P.D. 1986. Formulating multi-agent, dynamic-world problems in the classical planning framework. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82. Morgan Kaufmann, Los Altos, CA. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 675–710.

Pelavin, Richard N. 1991. Planning with simultaneous actions and external events. In *Reasoning about Plans*, pages 127–212. Morgan Kaufmann, San Mateo, CA.

Penberthy, J. Scott and Daniel S. Weld. 1992. UCPOP: A sound, complete, partial order planner for ADL. In Bernard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 103–114. Morgan Kaufmann, Boston, MA.

Pinto, Javier. 1994. *Temporal Reasoning in the Situation Calculus.* Ph.D. thesis, University of Toronto, Toronto, Ontario, Canada.

Poesio, Massimo. 1993a. Assigning a semantic scope to operators. In *Proceedings of the Thirty-first Annual Meeting of the Association for Computational Linguistics (ACL-93)*. Columbus, OH.

Poesio, Massimo. 1993b. A situation-theoretic formalization of definite description interpretation in plan elaboration dialogues. In P. Aczel, D. Israel, Y. Katagiri, and S. Peters, editors, *Situations Theory and its Applications, vol.3*, chapter 12, pages 343–378. CSLI, Stanford.

Poesio, Massimo. 1994. *Discourse Interpretation and the Scope of Operators*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. Available as Technical Report 518.

Poesio, Massimo, George Ferguson, Peter Heeman, Chung Hee Hwang, David R. Traum, James F. Allen, Nathaniel G. Martin, and Lenhart K. Schubert. 1994. Knowledge representation in the TRAINS system. To be presented at AAAI 1994 Fall Symposium on Knowledge Representation for Natural Language Processing in Implemented Systems.

Pollack, Martha E. 1986a. *Inferring domain plans in question-answering*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Pollack, Martha E. 1986b. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the Twenty-Fourth Annual Meeting of the Association for Computational Linguistics (ACL-86)*, pages 207–214. Columbia University, New York, NY.

Pollack, Martha E. 1990. Plans as complex mental attitudes. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press.

Pollack, Martha E. 1992. The uses of plans. *Artificial Intelligence*, 57. Revised transcription of the Computers and Thought Award lecture delivered at the Twelth International Joint Conference on Artifical Intelligence (IJCAI-91), August 26, 1991, in Sydney, Australia.

Pollock, John L. 1987. Defeasible reasoning. *Cognitive Science*, 11:481–518.

Pollock, John L. 1991. Self-defeating arguments. *Minds and Machines*, 1:367–392.

Pollock, John L. 1992a. How to reason defeasibly. *Artificial Intelligence*, 57:1–42.

Pollock, John L. 1992b. New foundations for practical reasoning. *Minds and Machines*, 2:113–144.

Poole, D. 1988. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–48.

Poole, D.L., R. Goebel, and R. Aleliunas. 1987. Theorist: a logical reasoning system for defaults and diagnosis. In N.J. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer-Verlag, New York.

Ramshaw, Lance A. 1991. A three-level model for plan exploration. In *Proceedings of the Twenty-Ninth Annual Meeting of the Association for Computational Linguistics (ACL-91)*. University of California, Berkeley, CA.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132.

Reiter, Raymond. 1992. The projection problem in the situation calculus: A soundness and completeness result, with an application to database updates. In *Proceedings of the First International Conference on AI Planning Systems*, pages 198–203. Morgan Kaufmann, College Park, MD.

Rescher, Nicholas. 1976. *Plausible Reasoning*. van Gorcum Publishing, Amsterdam, The Netherlands.

Rescher, Nicholas. 1977. *Dialectics*. State University of New York Press, Albany, NY.

Sacerdoti, Earl D. 1975. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214. Tbilisi, Georgia, USSR. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 162–170.

Sacerdoti, E.D. 1977. *A Structure for Plans and Behaviour*. Elsevier, North-Holland, New York, NY.

Sandewall, Erik. 1994. *Features and Fluents*. Oxford University Press.

Schubert, Lenhart. 1990. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Henry E. Kyburg, Jr., Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–68. Kluwer Academic Publishers, Dordrecht, The Netherlands. Also available as University of Rochester TR 306, August 1989.

Schubert, Lenhart K. 1992. Explanation closure, action closure, and the Sandewall test suite for reasoning about change. Technical Report 440, Department of Computer Science, University of Rochester, Rochester, NY.

Searle, John R. 1969. *Speech Acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, England.

Shanahan, Murray. 1989. Prediction is deduction but explanation is abduction. In Natesa Sastri Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1055–1060. Detroit, MI.

Shanahan, Murray. 1990. Representing continuous change in the situation calculus. In Luigia Carlucci Aiello, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, pages 598–603. Pitman Publishing Co., Stockholm, Sweden.

Shoham, Yoav. 1987. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104.

Shoham, Yoav. 1988. *Reasoning about change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, Cambridge, MA.

Simari, G.R. and R.P. Loui. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2–3).

Simon, Herbert A. 1977. *Models of Discovery*. Reidel, Dordrecht.

Sprites, P., G. Glymour, and R. Scheines. 1990. Causality from probability. In *Evolving Knowledge in Natural Science and Artificial Intelligence*, pages 181–199.

Tate, Austin. 1977. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–889. MIT, Cambridge, MA. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 291–296.

Toulmin, Stephen. 1958. *The Uses of Argument*. Cambridge University Press, Cambridge, England.

Traum, David R. 1994. *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester, NY. To appear as a Technical Report.

Traum, David R. and James F. Allen. 1994. Discourse obligations in dialogue processing. In *Proceedings of the Thirty-second Annual Meeting of the Association for Computational Linguistics (ACL-94)*.

Traum, David R. and Elizabeth A. Hinkelman. 1992. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8(3):575–599. Also available as University of Rochester TR 425.

van Beek, Peter, Robin Cohen, and Ken Schmidt. 1993. From plan critquing to clarification dialogues for cooperative response generation. *Computational Intelligence*, 9(2):132–154.

van Benthem, Johan F. A. K. 1983. *The Logic of Time*. D. Reidel and Kluwer, Dordrecht and Boston.

Vendler, Zeno. 1967. *Linguistics in Philosophy*. Cornell University Press, New York.

Vere, Stephen A. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 297–318.

Vilain, Marc and Henry Kautz. 1986. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 377–382. University of Pennsylvania, Philadelphia, PA.

Vilain, Marc, Henry Kautz, and Peter van Beek. 1990. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufman, San Mateo, CA.

Walton, Douglas N. 1992. *Plausible Argument in Everyday Conversation.* State University of New York Press, Albany, NY.

Wilkins, David E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA.

Yampratoom, Ed. 1994. Using simulation-based projection to plan in an uncertain and temporally complex world. Technical Report 531, Department of Computer Science, University of Rochester, Rochester, NY.

Yang, Qiang and Josh Tenenberg. 1990. ABTWEAK: Abstracting a non-linear, least commitment planner. In *Proceedings of the Eighth National Conference on Artificial Int elligence (AAAI-90)*, pages 204–209. Boston, MA.

A Knowledge Representation Functions

This appendix describes the abstract interface to the underlying knowledge representation system. While Rhet provides some of these services directly, this interface allows a different representational substrate to be plugged in without affecting the rest of the domain plan reasoner. It also serves as a definition of what capabilities are needed to support mixed-initiative plan reasoning of the complexity of the TRAINS domain plan reasoner.

A.1 Terms

`term-p` *x*

Returns true if *x* is a term.

`term-create &rest` *args*

Return the term corresponding to head and args in *args*.

`term-list` *term*

Return a list whose first element is the head of *term* and the rest of which is its arguments.

`term-nth` *n term*

Return the nth argument of *term* (the head is arg 0).

`term-head` *term*

Return the head functor of *term*.

`term-args` *term*

Return the arguments of *term*.

`term-bound-p` *term*

Return true if *term* contains no variables at any nesting.

A.2 Variables

`variable-create` *name* `&optional (`*type* `T-Anything)`

Return a new variable with name *name* and optional type *type*.

`variable-p` *x*

Returns true if *x* is a variable.

A.3 Lambda Expressions

`lambda-create` *var constraints*

Returns a lambda-expression containing *var* and *constraints*.

`is-lambda-p` *term*

Returns true if *term* is a lambda-expression.

`lambda-var` *term*

Returns the variable in lambda-term *term*.

`lambda-constraints` *term*

Returns the constraints in lambda-term *term*.

`lambda-skolemize` *lexpr*

Instantiate lambda event expression *lexpr* to a new constant. This means creating a new constant of the right type (may assert constraints if the type is structured) and then, if successful, asserting the instantiated constraints of *lexpr*. Return the new constant if successful, else `NIL`.

A.4 Formulas

`wff-p` *x*

Return true if *x* is a wff.

`wff-create &rest` *args*

Return the wff corresponding to head and args in *args*.

`wff-list` *wff*

Return a list whose first element is the head of *wff* and the rest of which is its arguments.

`wff-nth` *n wff*

Return the nth argument of *wff* (the head is arg 0).

`wff-head` *wff*

Return the predicate of *wff*.

`wff-args` *wff*

Return the arguments of *wff*

`wff-bound-p` *wff*

Return true if *wff* contains no variables at any nesting.

`wff-replace-if` *test wff new*

Return a new wff which is identical to *wff* except that any elements for which *test* returns non-`NIL` are replaced by *new*.

A.5 Types

`type-get` *x*

Return the type of *x* if it's known, otherwise `T`.

`type-set` *type* `&rest` *objs*

Set the type of *objs* to *type*. Returns `NIL` if any of the *objs* can't have their type changed.

`type-geq-p` *type1 type2*

Return true iff *type1* is equal to or is a supertype of *type2*.

`type-disjoint-p` *type1 type2*

Return true iff *type1* is not equal to or a sub- or supertype of *type2*.

`type-leaf-p` *type*

Return true iff *type1* has no subtypes.

`type-skolemize` *type*

Return a new term of *type*.

`type-subtype-list` *type* `&key (`*recursive* `t)`

Returns a list of all immediate subtypes of *type* (all subtypes if keyword argument *recursive* is `T`, the default).

A.6 Events

`roles-of-event-type` *type* `&key (`*synonyms* `nil)`

Returns a list of the rolenames for event type *type*. Roles marked as synonyms are not returned unless keyword argument *synonyms* is `T`.

`effects-of-event-type` *type*

Returns a list of the effects for event type *type*, leaving variable `?SELF` unbound.

`enablers-of-event-type` *type*

Returns a list of the enablers for event type *type*, leaving variable `?SELF` unbound.

`generators-of-event-type` *type*

Returns a list of the generators for event type *type*, leaving variable `?SELF` unbound.

`roles-of-event` *event*

Returns a list of role terms for event *event*. Note that this is different from `roles-of-event-type` which returns just the names (*i.e.*, the `:R-*` form).

`effects-of-event` *event*

Returns a list of the effects for event *event* (*i.e.*, `?SELF` is bound to *event*).

`enablers-of-event` *event*

Returns a list of the enablers for event *event* (*i.e.*, `?SELF` is bound to *event*).

`generators-of-event` *event*

Returns a list of the generators of event *event* (*i.e.*, `?SELF` is bound to *event*). This function takes care to ensure that the constraints of any structured terms referred to in the generators slot are properly initialized.

`make-event` *type*

Return a new constant of *type*, asserting constraints if consistent (else return `NIL`).

`print-event` *event*

Returns multiple values with information about *event*. Currently implementation-dependent and not much use to anyone except that it's easier to remember than `RETRIEVE-DEF`.

`agent-of-event-is-us` *event*

Returns true if the system is the agent of *event* (assumed non-lexpr).

`copy-event-except-agent` *event*

Return a new event of the same type as *event*, where all roles of the two events are set equal except the agent role (and any roles known to be equal to the agent role).

A.7 Equality

`eq-class` x

Returns a list of all objects known to be equal to x.

`ineq-class` x

Returns a list of all objects known to be not equal to x.

`eq-add` *term1 term2*

Adds an equality between *term1* and *term2* if it is consistent to do so. Returns `T` if they're already equal, `NIL` if they're already known to be not equal, otherwise something meaningless, but non-`NIL`.

`eq-test` *term1 term2*

Returns true if *term1* and *term2* are known to be equal (or are Lisp `EQUAL`). Both terms must be ground or an error is signalled.

`ineq-add` *term1 term2*

Adds an inequality between *term1* and *term2* if it is consistent to do so. Returns `T` if they're already not equal, `NIL` if they're already known to be equal, otherwise something meaningless, but non-`NIL`.

`ineq-test` *term1 term2*

Returns true if *term1* and *term2* are known to be not equal.

A.8 Time

`time-reln-set` *t1 reln t2*

Set the temporal relation between *t1* and *t2* to *reln* if possible.

`time-reln-get` *t1 t2*

Return the possible temporal relations between *t1* and *t2*. This is either a list or `:all`.

`time-reln-test` *t1 reln t2*

Test that *t1* and *t2* can consistently have relation *reln* (a list).

A.9 Knowledge Base

`kr-assert &rest` *wffs*

Add *wffs* to the database. Return non-`NIL` if all *wffs* can be asserted.

`kr-consistent-p &rest` *wffs*

Returns non-`NIL` if *wffs* are consistent with the database. Guaranteed not to throw, and guarantees that subsequent `KR-ASSERT` won't throw.

`kr-assert-if-consistent &rest` *wffs*

Assert *wffs* to the database if it is consistent to do so, else return `NIL`.

`kr-prove` *wff*

Return true if *wff* is provable in *context*.

`kr-prove-all` *wff*

Return a list of all instances of *wff* that are provable.

A.10 Miscellaneous

`is-occurs-p` *wff*

If *wff* is an event occurance formula, return the event, else `NIL`.

`is-event-p` *term*

If *term* denotes an event, return it, else `NIL`.

`lf-exists-skolemize` *wff*

Skolemize *wff* of the form

`(LF-EXISTS` *var dm rest phi*`)`

by creating a new constant of the same type as *var*, asserting that it is equal to *dm* (if consistent), asserting *rest* with *var* replaced by the constant (if consistent), and returning *phi* with *var* replaced by the constant.

B Plan Reasoner Functions

This appendix describes the interface exported by the domain plan reasoner to the rest of the TRAINS system. This includes description of the data structures used to represent plan graphs and the functions that operate on them.

B.1 Plan Nodes

```
(defstruct (plan-node (:print-function print-plan-node))
  type                  ; :event, :fact
  goal-p                ; t if this is a goal node
  label                 ; event or proposition label
  mark)                 ; status of node (old, new, etc.)

(defstruct (plan-event-node (:include plan-node (type :event))))
(defstruct (plan-fact-node (:include plan-node (type :fact))))
```

B.2 Plan Links

```
(defstruct (plan-link (:print-function print-plan-link))
  type                  ; type of link
  node                  ; the node at the end of the link
  parent                ; the parent (node at other end)
  mark                  ; status of node (old, etc.)
  (index 0))            ; where this link came from

(defstruct (plan-enables-link
            (:include plan-link (type :enables))))
(defstruct (plan-achieves-link
            (:include plan-link (type :achieves))))
(defstruct (plan-generates-link
            (:include plan-link (type :generates))))
(defstruct (plan-supports-link
```

```
            (:include plan-link (type :supports))))
(defstruct (plan-end-link
            (:include plan-link (type :end))))
```

`plan-link-label` *link*

Returns the label of the node in *link*.

`copy-plan-link-and-node` *link*

Return a copy of *link* whose node has also been copied.

B.3 Plan Subgraphs

`make-plan-subgraph` *link* `&optional` *children*

Return a new plan-subgraph consisting of *link* with *children*.

`plan-subgraph-link` *subgraph*

Returns the plan-link at the root of *subgraph*.

`plan-subgraph-children` *subgraph*

Returns the list of plan-link children in *subgraph*.

`copy-plan-subgraph` *subgraph*

Return a fresh copy of *subgraph* including a new link and node. The labels of the nodes are shared though.

B.4 Plan Graphs

`make-plan-graph` `&rest` *subgraphs*

Return a new plan-graph consisting of *subgraphs*.

`copy-plan-graph` *graph*

Returns a fresh copy of *graph* that shares no structure (nodes, links, or conses). Actually, the node labels are shared, but since when we modify a node destructively we replace its label, rather than modify the label destructively, this should be okay.

`plan-graph-nodes` *graph*

Return a list of all the plan-nodes in *graph*.

`plan-graph-links` *graph*

Returns a list of all the plan-links in *graph*.

`plan-graph-sinks` *graph*

Returns a list of toplevel plan-nodes in *graph*.

`plan-graph-sources` *graph*

Returns a list of plan-nodes in *graph* that have no children.

`plan-graph-mark-old` *graph*

Mark all nodes and links in *graph* as old. Nondestructive.

`plan-graph-find-node` *graph node*

Returns the subgraph of *graph* whose root node's label is `EQUAL` to the label of *node*, if any, otherwise `NIL`.

`plan-graph-goal-nodes` *graph*

Returns a list of plan-nodes that are goals in *graph*.

`plan-graph-event-nodes` *graph*

Returns a list of plan-nodes that are events in *graph*.

`plan-graph-event-goal-nodes` *graph*

Returns a list of plan-nodes that are event goals in *graph*.

`plan-graph-goals` *graph*

Returns a list of formulas that are goals in *graph*.

`plan-graph-events` *graph*

Returns a list of events labelling the event nodes in *graph*.

`plan-graph-event-goals` *graph*

Returns a list of events labelling the goal nodes in *graph*.

B.5 Plan Paths

`plan-path-to-plan-graph` *path*

Convert a plan-path *path* into a plan-graph and return it.

`plan-path-find-node` *path node*

Returns the link containing *node* in *path*, or *node* if *node* is the head of *path*.

`plan-graph-merge-path` *graph path*

Returns a freshly-consed plan graph resulting from merging *path* into *graph*. If some prefix of *path* is part of *graph*, then the result will have the links from *path* replacing those from *graph*, and the remaining links, if any, as children. If *path* doesn't appear at all, then it will be added as a new top-level subgraph of *graph*.

B.6 Plan Graph Searching

`init-plan-graph` *graph*

Initializes *graph* prior to searching it. This involves expanding each node once to create the initial search frontier, but only adding those children that are not already present. Returns the new graph.

`expand-plan-graph` *graph*

Expands *graph* by adding the children of all leaf nodes. Returns a freshly consed graph (is not destructive, except some labels may be Skolemized).

`search-plan-graph` *graph test* `&optional (`*depth* `0)`

Search *graph* until *test* is satisfied. Returns whatever non-`NIL` value *test* returned when applied to *graph*. Checks *depth* against the global variable `*SEARCH-DEPTH-BOUND*`.

B.7 Miscellaneous

`achievers-of-fact` *fact*

Return a list of lambda-event expressions for events that achieve *fact*. Note that the constraints in the lexpr may not be satisfiable.

`enablees-of-fact` *fact*

Return a list of `(`*lexpr* `.` *index*`)` pairs where *lexpr* denotes an event that is enabled by *fact*. Note that the constraints in the lexpr may not be satisfiable!

B.8 Plan Reasoner Interface Functions

`plan-unify` *wff1 wff2*

Test whether *wff1* and *wff2* unify, possibly with assumptions. Returns two values: a flag indicating if the unification was sucessful and a list of assumptions (or `NIL`). Both objects must be sentences. If they are of the form `(:occurs` *E*`)`, then the arguments are passed to `PLAN-UNIFY-EVENT`.

`incorporate` *formula plan* `&key` *incorp-type context purpose focus bg temp-seq plan-graph*

Incorporate *formula* into *plan*. The following keyword args are defined:

incorp-type	A keyword like `:goal`, `:role-filler`, etc. Only `:goal` is meaningful currently.
context	The root of the subgraph that we should search.
purpose	A formula that should be incorporated first and the results used to restrict the incorporation of *formula*.
focus	An object that the sentence focuses on (currently unused).
bg	Background information (unused).
temp-seq	Implicit (?) temporal sequencing information (unused).
plan-graph	The current plan graph. If specified, the *plan* term won't be used to build up the plan graph (which is expensive).

Returns a list containing:

result	`:ok` or `:choice` or `NIL`.
pdps	Plan description predicates describing additions to the plan as a result of incorporation.
graph	The resulting plan graph.
context	A plan-node that can be used for subsequent focussing via the *context* parameter.

If *result* is `:choice`, then *pdps* is a list of choices. If *result* is `NIL` then all bets are off, but *pdps* might be helpful.

`elaborate-plan` *plan* `&key` *plan-graph*

Elaborate *plan*. The following keyword args are defined:

plan-graph	The current plan graph. If specified, the *plan* term won't be used to build up the plan graph (which is expensive).

Returns a list containing:

result `:ok` or `:choice` or `NIL`.
pdps Plan description predicates describing additions to the plan as a result of elaboration.

If *result* is `:ok` and *pdps* is `NIL`, then the plan is complete according to the plan reasoner.