September 1992

Intentions in Means-End Planning (Dissertation Proposal)

Christopher W. Geib
University of Pennsylvania

Intentions in Means-End Planning (Dissertation Proposal)

Abstract

This proposal discusses the use of the intentions of the actor in performing means-end reasoning. In doing so, it will show that preconditions and applicability conditions in existing systems are ill-defined and intrinsically encode situational information that prevents intentions from playing a role in the planning process. While the former problem can be fixed, the latter cannot. Therefore, I argue that preconditions should be eliminated from action representation. In their place, I suggest explicit representation of intention, situated reasoning about the results of action, and robust failure mechanisms. I then describe a system, the Intentional Planning System (ItPlanS), which embodies these ideas, compare ItPlanS to other systems, and propose future directions for this work.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-73.

Intentions in Means-Ends Planning (Dissertation Proposal)

MS-CIS-92-73
LINC LAB 235

Christopher W. Geib

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

September 1992

Intentions in Means-End Planning[1]

Dissertation Proposal

Christopher W. Geib

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA
geib@linc.cis.upenn.edu

[1]This research is supported by ARO grant DAAL03.89.C.0031 PRIME

Abstract

This proposal discusses the use of the intentions of the actor in performing means-end reasoning. In doing so, it will show that preconditions and applicability conditions in existing systems are ill-defined and intrinsically encode situational information that prevents intentions from playing a role in the planning process. While the former problem can be fixed, the latter cannot. Therefore, I argue that preconditions should be eliminated from action representation. In their place, I suggest explicit representation of intention, situated reasoning about the results of action, and robust failure mechanisms. I then describe a system, the Intentional Planning System (ItPlanS), which embodies these ideas, compare ItPlanS to other systems, and propose future directions for this work.

Contents

List of Figures

Chapter 1

Introduction

Consider the simple blocks-world situation shown in Figure 1.1 .

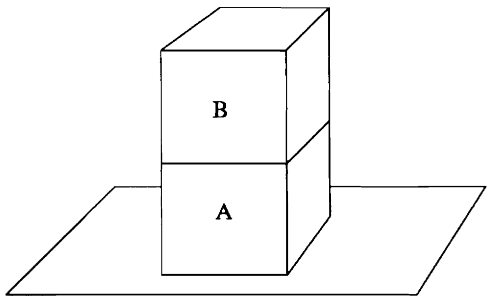

Figure 1.1: A simple blocks-world situation

In most planning systems, if the system is instructed to pick up block "A", it will first move block "B" to the table. If, on the other hand, a human agent is instructed to pick up block "A", there are three possibilities. First, the agent may, like the planner, move block "B" to the table, then grasp and lift block "A". Second, the agent may, by grasping block "A", lift both block "A" and block "B", achieving the desired goal. Third, the agent may grasp block "A" and, by pulling laterally and upward, pull block "A" from under block "B". Notice that, only in the first case does the agent actually remove block "B" before engaging in the desired task.

This behavior is very suggestive. Many planning systems have a precondition on the action "pickup" that the object to be lifted must be clear, and yet, in two of the obvious solutions to this directive, this precondition is not met. The question then is, why should the action "pickup" have a precondition that the block to be lifted be clear? In this paper I will argue that it shouldn't. I will argue that, in fact, preconditions as a whole should be eliminated in favor of explicit representation of intentions, situated reasoning about the effect of actions, and robust failure mechanisms.

The strength of two of these mechanisms can be seen in the previous situation. For example, let us assume that the agent's intentions are explicitly encoded. Situated reasoning is sufficient for the agent to determine which methods of lifting the blocks are acceptable and which are not. If the agent has no intentional commitments about the fate of block "B", it will not be constrained to "clear" block "A" before lifting. In this case, the agent might select between any of the methods of picking up the block. If, however, the agent has as one of its intentions that it not break objects, then pulling block "A" from under block "B" is not an admissible solution, since block "B" might fall and break. Of course, the solution of lifting both blocks is still viable, provided the agent takes care to prevent the block on top from sliding off. Finally, if the agent is very concerned about the fragility of block "B", there is only one solution that is admissible.

The view being proposed here, namely the use of intentions as a filter on the selection of various solutions to a problem is not new. As Bratman argues [4]

> My prior intentions and plans, then, pose problems for deliberation, thereby establishing standards of *relevance* for options considered in deliberation. And they constrain solutions to these problems, providing a *filter of admissibility* for options.

It is only by considering the network of those actions that we intend and those that we do not intend that we are able to arrive at correct decisions about methods of achieving our goals and the conditions that should hold before acting. It is through this use of intentions that people are capable of making decisions about actions with the "correct" results in variety of situations presented by the world every day.

1.1 Thesis Statement

Specifically this proposal argues for two points:

1.) *Preconditions have been used in existing systems to encode situation-dependent information about actions. Thus, preconditions limit the effective application of intentions to the means-ends reasoning involved in the planning process. In order for a planner to give intentions their correct role in the planning process, preconditions, as previously conceived, must be eliminated from action representations.*

2.) *By explicitly representing positive and negative intentions and using situated intentional reasoning and robust failure mechanisms, preconditions can be replaced without reintroducing the problems associated with them.*

To establish these claims, this paper will consider various definitions for preconditions and show the problems associated with each of them. To validate these claims, this paper will present a planner based on these arguments called the Intentional Planning System (ItPlanS), and outline how the system functions.

1.2 Outline of Proposal

Chapter 2 will present a general definition for preconditions. After pointing out inadequacies in other definitions, I will show how even this general definition of preconditions has a fundamental flaw. Chapter 3 of this paper will be devoted to defining the terms necessary for the my proposed solution to this problem. Chapter 4 will then discuss how intentions, situated reasoning and robust failure mechanisms can be used to overcome the problems that preconditions pose.

Chapter 5 will describe the Intentional Planning System (ItPlanS) as a functioning instantiation of the ideas outlined in Chapter 4, including a discussion of the data structures used and an outline of the system algorithm. Chapter 6 will discuss the similarities and differences between ItPlanS and other existing systems. Finally, conclusions and directions for proposed future work will be covered in Chapter 7.

Chapter 2

Preconditions and Their Problems

The thesis of this work is that preconditions limit the effective application of intentions to the planning process. In this chapter I will discuss a definition for preconditions and argue for its coverage. I will then discuss how other system's definitions are subsumed by this definition, and finally, explain how even this general formulation of preconditions interferes with the planning process.

2.1 Definition

The definition of preconditions I will use was first put forward by Pollack [23]. She defines preconditions in terms of Goldman's generation relation. In the following section, I will define the generation relation, present Pollack's arguments about the definition of preconditions, and show how her definition subsumes other definitions and uses.

2.1.1 Defining Terms: Goldman and Pollack

In his work, Goldman [14] defined a relation called the generation relation. This relationship holds between two actions when the occurrence of one constitutes an occurrence of the other. For example, the action "John moved his queen to king-knight-seven" generates "John checkmated his opponent" [14, page 40]. The following is Goldman's original definition for generation.

Act-token α level-generates act-token β if and only if

1. α and β are distinct act-tokens of the same agent that are not on the same level;
2. neither α nor β is subsequent to the other; neither α nor β is a temporal part of the other; and α and β are not co-temporal;

3. there is a set of conditions C such that
   (a) the conjunction of α and C entails β, but neither α nor C alone entails β;
   (b) if the agent had not done α, then he would not have done β
   (c) if C had not obtained, then even though G did α, he would not have done β. [14, page 43]

While Goldman describes multiple types of generation, these distinctions will not be relevant here. Therefore, in the following discussion, I will use the term generation rather than level-generation or any of their more specific types of generation.

Pollack formalized Goldman's definition using predicates defined by Allen [3]. To represent the fact that a given predicate holds at a given time she uses the *HOLDS* predicate, and to represent that an action occurs, the *OCCURS* predicate. Thus Pollack formalizes Goldman's definition of generation as the *CGEN* relation with the following definition:

$$\begin{aligned}&CGEN(\alpha,\beta,C) \longleftrightarrow \\ &\quad \forall G \forall t_1[[HOLDS(C,t_1)\ \wedge\ OCCURS(\alpha,G,t_1)] \rightarrow OCCURS(\beta,G,t_1)] \wedge \\ &\quad \exists G\, \exists t_2[OCCURS(\alpha,G,t_2)\ \wedge\ \neg OCCURS(\beta,G,t_2)] \wedge \\ &\quad \exists G \exists t_3[HOLDS(C,t_3)\ \wedge\ \neg OCCURS(\beta,G,t_3)]\ \text{[23, page 51]}\end{aligned}$$

where α and β are actions, G is an agent, C represents the generation conditions, and t_1 and t_2 are times.

Pollack then defines a predicate *EXEC*, in terms of Goldman's basic actions and standard conditions for actions. (A complete definition for this predicate can be found in [23].) It is sufficient to know that the *EXEC* relation defines those conditions in which an agent is physically able to perform some action. Thus, $EXEC(\alpha,G,t)$ is true if and only if there are no external forces which would prevent agent G from performing action α at time t.

Using these definitions, Pollack defines the relationship between preconditions and actions. Consider a generic action represented by a header (β), a body (α), and a precondition formula C. Pollack defines C as a set of preconditions on performing β by doing α if and only if they allow α to generate β and they guarantee the executability of α. Thus, the precondition relation is defined in the following way.

$$PRECOND(C,\beta,\alpha) \longleftrightarrow CGEN(\alpha,\beta,C) \wedge [HOLDS(C,t) \rightarrow \forall G\ EXEC(\alpha,G,t)]$$

That is, preconditions are those conditions that allow the execution of the action α to generate an occurrence of β and are also sufficient to guarantee that there is no physical impediment to the execution of α. For the rest of this paper, I will refer to conditions that allow an action to generate another action as *generation conditions* and those conditions that guarantee that an action is executable, as *executability conditions.*

As Pollack argues, it is not possible to use either clause alone for a definition. If the conditions C were only the generation conditions then they would not guarantee the executability of α and hence would not guarantee the generation of β. Conversely, if the conditions merely guaranteed the executability of α there would be no guarantee that the performance of α would generate β. Thus, according to Pollack, neither of these conditions alone are enough. Only the conjunction will suffice.

2.1.2 Existing System Definitions of Preconditions

As Pollack observes, "much of the existing planning literature has been vague about the intended interpretation of action operators, and has used them at different times to mean different things." [23, page 74] Therefore, in the following section I will consider some of the existing system definitions, show how they are inadequate, and then show how preconditions taken from these same systems fit within Pollack's definition.

STRIPS style

Preconditions are "...the conditions under which the operator[action] is applicable." [10, page 192]. This sentence, or one very much like it, appears as the definition for preconditions in many planning systems [10, 20, 25, 33]. Using this definition, preconditions define those states from which the given action can be taken. That is to say, unless the preconditions are met the action cannot even be attempted. As Lifschitz [20] formalizes this definition, an action/operator is only defined for those states in which the preconditions are true. This definition is obviously inadequate.

As Hanks points out, this definition "... confuses the notion of an action being *meaningful* or *conceivable* with the notion of an action's achieving its intended effect." [16, page 159] Looking at the example of picking up blocks, not only can the action of picking up a block be attempted in cases where the preconditions are not satisfied, but they can also be successfully executed in these cases.

While this definition for preconditions is obvious lacking, it is interesting to notice that STRIPS preconditions fall under the definition given by Pollack. For example, consider the following operator from [22, page 281]

pickup(x)

preconditions: ontable(x), clear(x), handempty

delete list: ontable(x), clear(x), handempty

add list: holding(x)

All three of the preconditions given for this operator are clearly conditions under which a pickup would be generated by the unspecified series of subactions that make up the body of this action.

ABSTRIPS

ABSTRIPS is an interesting system in terms of its preconditions. While the definition for preconditions follows STRIPS and the actual preconditions again fall into the categories Pollack describes, ABSTRIPS is one of the few systems to encode executability conditions. Consider the following set of preconditions for an action to push an object (bx) to another object (by).

> TYPE(by,object),PUSHABLE(bx),
> NEXTTO(ROBOT,bx), $\exists$ rx [INROOM(bx,rx) $\wedge$
> INROOM(by,rx) $\wedge$ INROOM(ROBOT,rx)] [25, page 125]

In this case, the precondition PUSHABLE(bx) is an executability condition. This is in sharp contrast with the majority of systems which assume that the agent is universally able to perform its actions. For example, [15, 21, 31, 33] don't encode the conditions under which the action is executable but rather only those conditions that generate the action.

Applicability Conditions

In his work on NONLIN [30], Tate has defined what he refers to as *usewhen conditions.* These conditions correspond to what Georgeff has referred to as *triggering* conditions or what have been referred to by Schoppers as *applicability* conditions. Tate defines them by saying "Conditions may be stated which must hold before this expert[action] can be called into use at all." [30, page 293]. This definition differs little from the one specified in STRIPS; however, if the operational semantics of the condition is examined, differences between these conditions and standard STRIPS preconditions can be seen.

In NONLIN, the agent does not attempt to make usewhen conditions true before using the action. If the conditions are not true, then the action is considered inappropriate or not applicable. For example, a usewhen condition on the action of unplugging an object, would be that the object be plugged in to begin with.

I believe that an action's applicability to a situation is determined by its possible results. If its execution may satisfy the agent's intentions or goals, then it is applicable. If it does not satisfy the agent's current intention, then it is not applicable. For this reason, these conditions will be ignored in the following work. Interestingly however, these conditions are still subsumed by Pollack's definition within the executability conditions. For example, the PUSHABLE condition from the last example might be considered to be an applicability condition.

2.2 The Problem with Preconditions

Even Pollack's definition for preconditions has a problem. Her definition assumes that the agent's goal is the successful performance of the action given in the header. However, if as suggested in the previous section, an action is considered relevant if its effects achieve the agent's goals, this assumption is unwarranted.

For example, consider the following hypothetical action.

Header: unstack(X,Y)

Preconditions: P

Body: moveto(X), grasp(X), lift(X), moveto(table), release(X)

Effects: on(X,table), handempty, clear(Y)

Assume that the precondition is some set of conditions that falls within Pollack's definition. One of the effects of this action is that X is no longer on Y, but this is not the action's only effect. Another effect is that Y is clear, and this effect can be achieved even if "unstack" is not completed successfully.

Thus, if an agent were using an unstack action in order to clear some object, the given preconditions would be misleading, since they are based on the assumption that the complete unstack operation needs to be performed successfully. If we assume that Y is cleared after the lift action, then the final two subactions of the unstack operation can succeed or fail. Their success or failure makes no difference for the agent's goal, and therefore, the preconditions that have to do with these last two actions can be eliminated. This simple example demonstrates that the preconditions of any action are dependent on the goal the action is being used to satisfy, which may and or may not be specified by the header. Thus, a usage-independent determination of preconditions is impossible.

One might question why an agent would choose to use an unstack action to clear a block, especially since the agent must be able to accomplish the relevant series of subactions. There are a number of reasons for such a choice. First and most obviously, this action might be the only single action the planner has at its disposal that accomplishes the task. Second, the planner might not be aware that a sequence of actions would have the desired effect. For example, in this case the planner might not be aware that simply grasping the object and lifting would achieve its goals.

Finally, and perhaps most relevant to this thesis, an agent might be using an action to accomplish more than one intention. For example, suppose an agent has three goals: clearing a block, having the robot's hand at table level, and having the robot's hand empty. While a successful unstack action will accomplish all three of these tasks, it is not necessary that each of the unstack actions subactions be successful in order to achieve all of these goals: the block might slip out of the robot's hand and break. However, since this is unrelated to the system's goals, preconditions that would prevent it would not need to be enforced before taking the action.

In summary, the preconditions of an action depend crucially on the environment in which the action is undertaken and the goals it is invoked to achieve. There is no way in which this context sensitivity can be eliminated. Therefore, a principled treatment of intentions in means-ends reasoning in planning, requires that preconditions be eliminated from action representation and replaced by situated intentional reasoning to derive those conditions that are significant.

Chapter 3

Defining Intentions

I have proposed that preconditions should be replaced by "explicit representation of intentions and situated intentional reasoning." In this chapter I will define what I mean by intention. This definition is not intended to be a complete, formal definition, but rather it will ground the term for discussion so that an operational semantics can be designed. For more formal treatments of these issues the reader is referred to [4, 6, 7, 23]. I will follow Bratman [4] in defining intentions as a separate and identifiable mental state. This can be contrasted with the belief/desire model of intentions [8, 17], in which intentions are not atomic, but are composed out of beliefs and desires.

Unlike Bratman, who uniformly uses the term "intention" to denote any commitment toward an action (positive or negative), I have found it helpful to define three possible intentional states: positive, negative, and unintended. These distinctions should be seen as classifying intentions on the basis of how they control behavior rather than separating different kinds of intentions. A brief mention of how these intentions control behavior will be made here, and a more thorough treatment can be found in Chapters 4 and 5.

3.1 Positive Intentions

The definition of positive intentions is captured in the the common use of the word "intend" in sentences such as, "I intend to go to the store." or "I intend to eat lunch." A *positive intention* is a commitment by an agent to perform an action at a specific time. Thus, if an agent intends to perform some action, then assuming that it still has that intention when the time for action arrives, it will perform that act.

There is some confusion concerning agents intending states or intending "situations." As mentioned above, intentions are attitudes taken toward actions; therefore, it is impossible for an agent to directly intend a state. However, it is possible for an agent to intend to act in a way that will bring about a state. Thus, the agent intends to perform an unspecified action that will cause the desired state. Notice however, the agent's commitment is still to the action, not the state.

This kind of goal-directed intention does not imply that there exists an action that will cause the goal to be true or that the agent would intend such an action if it does exist. For example, an agent can intend to "become rich." According to our definition of intending a state this would translate into the agent intending to perform some action that will cause the agent to be rich. However, there may not exist any action with this result. An agent can, therefore, intend to achieve a state that it has no way of achieving. This problem is not new; see [6] for further discussion.

3.2 Negative Intentions

In contrast to positive intentions, *negative intentions* describe those actions an agent is committed to NOT executing. Thus, negative intentions allow the agent to specifically eliminate certain actions from deliberation. However, it is not intuitively obvious what the agent is committed to when its only commitment is to NOT act in a specified manner. In fact, negative intentions tell us nothing about what the agent WILL do; instead they enumerate those actions that the agent is committed to not performing.

As with positive intentions, the commitment an agent makes when forming a negative intention is to an action. Therefore, it is impossible for an agent to directly negatively intend a state. However, as with positive intentions, it is possible for an agent to negatively intend any action it believes will causes a particular state.

Thus, in negatively intending a state, the agent intends to not perform any action it believes will cause the state. This gives negative intentions considerable power. For example, a typical negative intention might be **BROKEN(obj1)** where **obj1** is a variable. Having this as a negative intention would prevent an agent from performing any action it believes would result in any object (that can be bound to **obj1**) being broken.

Notice that again the negatively intending a state does not commit to the existence of an action that achieves the goal. Therefore, as with positive intentions, negative intentions do not commit to the existence of such an action, but merely prevent the agent from taking it if it exists.

Preventive Action

The given definition of negative intention must be distinguished from another stronger possible definition. Negatively intending a state could be defined as requiring that the agent intends that the world not be in the specified state at the given time. The principal difference between the two is in the strength of the commitment made by the agent. In the original definition, the agent is committed to not performing an action that will cause the state, while the second commits the agent to not allowing

ANY AGENT (animate or inanimate) to cause the state.

In short, the second definition licenses preventive action on the part of the agent, while the first does not. Since this work assumes that the agent in question is never the only agent of change in the world, agents will engage in action to prevent interference by inanimate forces. However, the issues raised by having an agent engage in behaviors designed to prevent interference by intelligent agents is well beyond the scope of this work. Therefore, in this work the first definition has been adopted.

3.3 Unintended Actions

For the purposes of this work, all actions or classes of actions that are not specifically positively or negatively intended are said to be *unintended.* As we will see, an action may start out being unintended; however, if the action is added to a plan, the agent positively intends that action until the action is removed from the plan.

Chapter 4

Solutions to the Problems of Preconditions

Chapter 2 described the problem associated with the use of preconditions in action representation. However, since preconditions do perform important functions, it is not sufficient to simply eliminate them; their function must be replaced. In this chapter, I will show how the use of explicit representation of intentions, situated reasoning, and robust failure mechanisms can replace preconditions without encoding situation-specific information.

4.1 Explicit Representation of Intentions

Almost all planning systems use an explicit representation of the system's goals. As a result, these systems are already representing some of their positive state-based intentions. This explicit representation of intentions must be extended to include positive intentions about actions and negative intentions as well.

While the benefits of extending the explicit representation of positive intentions to include intentions toward specific actions are obvious, the benefits of explicit representation of negative intentions are unclear. I will show that negative intentions are required for an agent to determine if an action has been successful.

It is my position that there should be no pre-defined goal that an action is designed to achieve. Unlike [11] and [33], an agent should be able to use an action to achieve any state that results from its execution. Thus, the success of an action must now be a function of an agent's positive and negative intentions. As a result, an agent's negative intentions must be explicitly represented. By examining its positive intentions, an agent can determine if an action has achieved its intended effect, and by examining its negative intentions, the agent can check for undesired effects. Notice that if the desired and undesired effects of an action are defined relative to the intentions of the agent, they are situation-dependent. Since an action may be called upon to achieve very different intentions in different situations, this is exactly the desired result.

Finally, since positive and negative intentions serve to define the desired and undesired effects of an action, they will also be of primary importance in performing the situated reasoning necessary for the selection of actions. Thus, they are necessary not only to determine the success or failure of an action, but also to determine which actions should be taken.

4.2 Situated Reasoning

An argument has been put forward in Chapter 2 for the use of *situated reasoning.* I will use this term to denote reasoning about the effects of performing an action in a given world state. This kind of reasoning is integral to the process of determining appropriate actions. This section will discuss different kinds of reasoning about the effects of action, show why this reasoning requires being situated in a world state, and demonstrate how it may be used to select actions.

As we have seen, agents must consider their intentions in order to make decisions about the desirability of a given action in a given situation. However, this is not the only situational piece of information needed for reasoning about the effects of actions. Often an agent's own *memory* will be an important factor in adopting intentions to act. Frequently the only evidence an agent has of performing an action is its memory. For example, it is unlikely that there will be any physical evidence of jumping into the air. Therefore, the only way in which the agent could be aware that it had satisfied an intention of this kind would be to consult its memory.

Accurate reasoning about the results of action also requires the reasoner to perform physical simulation of the action. Since the reasoning must remain as close to the actual world as possible, the agent must consider, not an idealized model, but rather a specific, accurate model of the actual objects. Since maintaining a model of the entire world with the requisite detail would be impossible, this reasoning cannot be performed on a complete model. When models of the world are required for the reasoning, they must be small and tailored to the action and situation in question.

This kind of detailed reasoning about the results of actions can be used for two purposes in a planning system. First, action reasoning can be done to prevent violating negative intentions; before the agent carries out an action, it can reason to ensure that the action will not violate one of its negative intentions. In short, an agent can use its positive intentions to select a possible action and then reason about the results of the action, given the current world state, to verify that the positive intentions of the agent are achieved and that none of the negative intentions are violated.

Second, an agent can use small tailored models of the world to reason about achieving multiple intentions. Any pair of intentions may interact in several ways. Rational agents should be able to identify and utilize positive interactions between the methods of achieving their intentions. For example, an agent should, when possible, choose methods of achieving current intentions that accomplish parts of future intentions. An agent must also be able to realize and deal with conflicts between negative and positive intentions. For example, if an agent has only one way of achieving a given intention and that method violates one of the agent's negative intentions, a rational agent needs to recognize this conflict and attempt to avoid it. Conflicts can also occur between two positive intentions, as for example, in the well known Sussman anomaly [29].

While these conflicts might be foreseen outside of the situated environment, it may be too costly to consider all of the possible interactions that might occur at run-time. Many of these conflicts will not, in fact, come into being. A given intention may be satisfied by other agents, or the agent may be forced to drop one of its intentions. Thus, for many of the same reasons that it is unrealistic to create complete plans before engaging the actual world, it is undesirable to plan for all of the possible conflicts between intentions outside of the situation in which they occur. It is preferable to spend only those resources necessary to resolve the interactions that arise in the course of action with situated reasoning.

4.3 Robust Failure Mechanisms

Before discussing failure mechanisms, I must define what it means for an action to fail in this context. Since the goal of any action is to achieve a specific intention, a *failed* action is defined as an action that, when actually performed, does not achieve its intended goal. It does not matter if the action was performed correctly but failed to have the intended effect or if the action was performed incorrectly; both cases are action failures. Thus, action failure can only be detected after the action has been performed. Notice that an action has not failed if it achieves its intention and violates a negative intention. This situation is undesirable but not considered an action failure.

Now a brief word must be said about why failure mechanisms should be considered as an important part of a replacement for preconditions. Since I have taken the position that actions do not have pre-defined goals and preconditions must be removed from action representations, the definition of an action no longer prevents its use at inappropriate times. An agent can now attempt any action at any time. Unfortunately, most actions will not achieve their desired goals in every possible world, and many actions will have nonobvious reasons for failure. It would be unreasonable to expect that a situated reasoner would be able to foresee all of these conditions before the action is taken. In fact, in the system I describe in Chapter 5, after the agent has committed to performing an action, no attempt is made to derive possible causes of failure, before the action is undertaken. Only when the action fails to achieve the intended effect are causes of failure searched for. Therefore, more than other planning formulations, a system based on these principles will have to confront the issue of action failure.

Given the likelihood of an action's failure, there are two possible courses of action an agent might take. First, the agent might determine that the plan it was using was incorrect. In such a case, the agent would re-plan how to achieve the goal it was pursuing. Second, rather than re-planning, the agent might eliminate the condition that caused the action to fail. The difference is quite simple; in the case of re-planning, the planner abandons its previous plan in favor of a new method. In action repair, on the other hand, the planner takes on a new goal, namely eliminating conditions that prevent the successful performance of the action.

4.3.1 Re-planning

Re-planning is a special case of the general planning problem. Once the agent has determined that an action is not achieving its intended effect, it can reconsider its method of achieving the goal. Since situated reasoning must involve the agent's memory, the agent will be aware that the first method failed to have the desired result and can choose another method.

The more significant issue in re-planning is to determine when it is more profitable for the agent to discard the existing plan and start again. There are a number of relevant factors: the amount of planning already accomplished, the desirability of side effects of the original plan, and the number of completed actions in the plan. In the end, the question is complex and largely subjective; I will not attempt to answer it here.

4.3.2 Action Repair

In action repair, unlike re-planning, the method for achieving the goal is not altered, but rather, the agent acts to alter the world state so that the condition preventing the successful execution of the action is removed. Then the action is retried. A typical example of such a condition would be a potato in the tail pipe of a car preventing the car from starting.[1] In general the agent would simply attempt to start the car. However, when turning the key fails, the agent does not have another plan to start the car. Therefore, it must find the problem with the existing plan and eliminate it (in this case the potato).

A general, robust, information-independent strategy for the process of identifying conditions of this type is desirable. Unfortunately, the only general strategy for this problem is to appeal to someone who knows more about the failure than you do. Notice however, that people do not spend all day asking other people to solve their problems for them. In fact, for the vast majority of the action failures, once a person realizes the action is failing, they correct the problem causing the failure and resume

[1]While the example is the same as that used by McCarthy in his work on circumscription, this treatment has nothing to do with circumscription.

the action. In fact, they may correct several "errors" before eliminating the actual cause of the problem.

Despite the varied and often difficult domains in which problems occur, people are able to solve them, because they have, as Hammond [15] points out, information about the causes of previous failures. By remembering these causes and their solutions, an agent will know more the next time a similar failure occurs. Simply put, if an agent is familiar with the common problems in starting a car (and their solutions), and if a car doesn't start, then the agent may be able to solve the problem.

This provides a domain-independent, robust action repair mechanism. However, this mechanism is not knowledge-independent. In fact, it is not only knowledge-dependent but knowledge-intensive. The agent must know those conditions that are likely to cause an action to fail, and how to correct them. Only if the agent has both of these pieces of information, will it be able to recover from the failure of an action.

There is an obvious question to be asked. If the agent has this list of possible causes of failure, why not check this list before the action is taken? There are a number of reasons. First, these are conditions are *possible* causes of failure, not definite causes. This means that, even if the condition is true, the action may still succeed. For example, the car may start and the potato may be blown out of the tail pipe. Second, the list of possible causes of failure may be quite long (possibly infinite). If an agent were to check all of the possible reasons for a car not to start, the agent might never manage to start the car. Therefore, while an agent must have this information, it only need consult it after the action has failed.

Both re-planning and action repair are required for any planning system that attempts to operate in real domains. In Chapter 6, more contrast will be drawn between re-planning and action repair, and some perspective will also be given on other solutions to these problems. All of these methods will be used in the system that will be described in the next chapter.

Chapter 5

ItPlanS

The Intentional Planning System (ItPlanS) embodies the ideas put forth in the previous chapters. It uses no preconditions in its action representations. Due to this absence, ItPlanS cannot use traditional backchaining methods. Therefore, it is designed as a hierarchical planner, and in many ways resembles other hierarchical planners [11, 13, 21, 25].

ItPlanS starts by creating an intentional structure to represent the agent's commitments to act. It is through the expansion of these intentions that planning takes place and more intentional commitments are made by the agent. However, this expansion only takes place to the degree required for the agent to begin to act. This chapter will explain the functioning of the system by describing various system specific data structures, followed by a detailed description of the system's algorithm. Chapter 6 will then compare the system with other existing planners.

As a side note, ItPlanS has been used in two different domains: a blocks-world in which the agent has two effectors and a human task domain in the AnimNL project [32] at the University of Pennsylvania. Since the blocks world is a simple domain to understand, most of the examples used in this chapter are taken from that domain.

5.1 Intention Data Structures

Before beginning a detailed discussion of the planner's algorithm, it is necessary to describe the data structures used to implement various parts of the theory. ItPlanS has two kinds of intentions, positive and negative, and since they are used in different ways, they are represented in separate data structures.

5.1.1 Negative Intentions

Negative intentions are represented as a list of actions and goals, which represent the actions the agent is committed to not performing and the states it is committed to avoid. Since the commitment to negative intentions is not order-dependent, a list is a sufficient data structure for them. In processing, the system will merely need to know if any of the negative intentions might be violated by performing an action.

5.1.2 Positive Intentions

In contrast to the simple data structure of the negative intentions, the data structure required for positive intentions is quite complex. The system's representation begins with the assumption that the intentions given it are completely temporally ordered. The order is assumed to be significant and fixed. Note the system will not deal with non-specific meta-intentions. For example, the system can not handle an intention to "be careful."

Within the intention list, each intention is represented by a five-tuple that contains: the intention itself, a unique identifier, an indicator of the type of intention it is (maintained or achievement), a number representing the expansion of the action being used to satisfy the intention (actions can have multiple possible expansions), and the actual action being used to satisfy the intention. Each action is made up of three parts: an action, a unique identifier, and a possibly empty list of sub-intentions. Obviously this structure is highly recursive. Example 1 represents an intention to achieve on(a,b), by performing a stack operation:

Ex. 1 *intend(on(a,b), 3, maint, 1, action(stack(a,b), 4, [intend(...), intend(...)...]))*

The roles of the first and second arguments in this structure are obvious. The third argument reflects the fact that ItPlanS distinguishes between two "types" of positive intentions: maintained (maint) and achievement (ach). When ItPlanS was first constructed, all intentions were of the achievement variety. However, there are often conditions that, once achieved, should hold throughout the performance of the rest of an action, for example, holding an object while carrying it to another location. If when traveling, the agent loses control of the object, the agent should stop and regain control before continuing. Maintained intentions were included in the system to account for these sorts of conditions. To the degree that preconditions are also sometimes used for this function, maintained intentions serve to replace them.

Maintained and achievement intentions differ principally in how they are treated after they are accomplished. An achievement intention, once satisfied, is removed from the agent's intention structure. For example, suppose an agent an intention to jump over a small creek. This would be considered an achievement intention. Once the agent has jumped the creek, the intention has been achieved and the agent no longer considers it. It can safely be removed from the agent's intention structure.

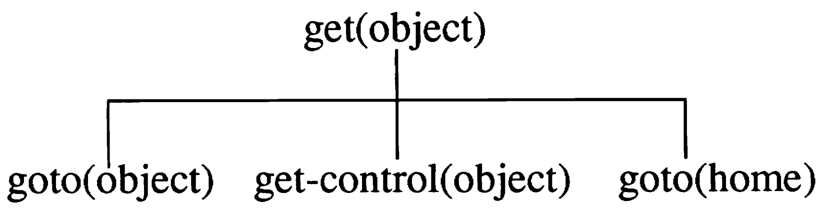

Figure 5.1: Simple intention decomposition of get

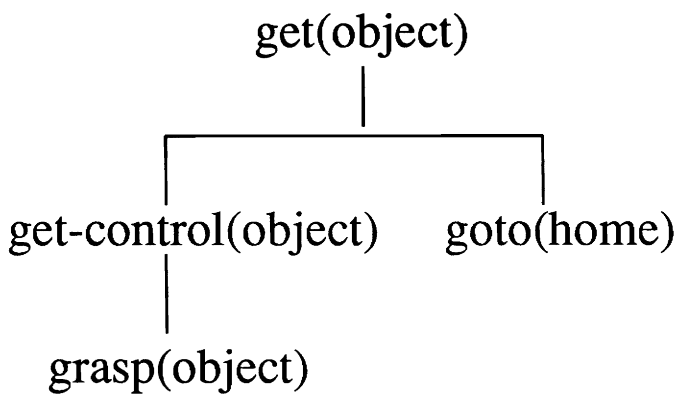

Figure 5.2: Decomposition of get control intention

On the other hand, once a maintained intention has been achieved, all of its subintentions are removed from the structure, but the intention itself is retained in the intention structure until its parent intention has been achieved. For example, suppose an agent has the intention to get an object and return to its original location. ItPlanS would break the "get" intention down into three subintentions as shown in Figure 5.1: going to the object, getting control of the object, and returning to the starting position. Let us assume that the first intention of going to the object is an achievement intention and that the get-control intention is a maintained intention.

Once the agent is at the object, the intention of going to it will be removed from the the structure. Figure 5.2 shows the intention structure while the agent is achieving the get-control intention, and Figure 5.3 shows the agent's intention structure while the agent is returning. Notice that in Figure 5.3, while the expansion of the get-control intention has been removed, the intention itself is still in the structure. The agent still has the intention to have control of the object. This allows the agent to monitor the status of this intention, and if something happens to violate it, take action to re-achieve it.

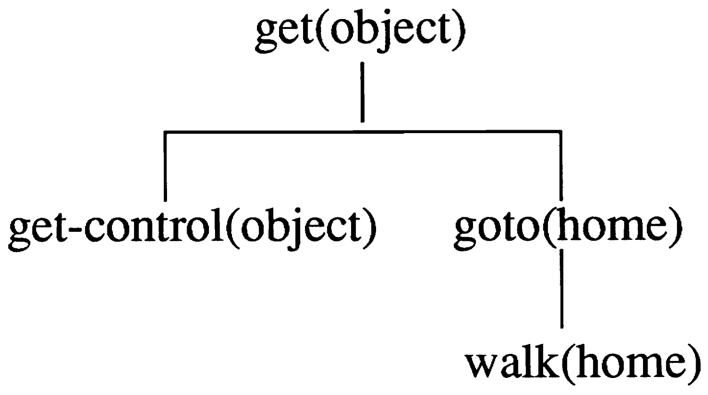

Figure 5.3: Intention structure when returning

5.2 Action Representation

Since ItPlanS is a hierarchical planner, it has two distinct types of action in its ontology: primitive and complex. *Primitive actions* are actions in the traditional sense. They have a physical realization; they perform some tangible operation on the world. They can be thought of as the system's "basic actions" [14, 23]. *Complex actions*, on the other hand, perform the hierarchical decomposition of the system's intentions. The following sections will define each of these types of action and explain how the system uses them.

5.2.1 Primitive Actions

As noted, primitive actions are the simplest operations the system is capable of performing on the world. In the case of the blocks-world domain, I have assumed that a two-handed robot has six of these: RIGHTGRASP, LEFTGRASP, RIGHTRELEASE, LEFTRELEASE, RIGHTGOTO, and LEFTGOTO. The selection of these actions and this number is arbitrary.

The grasp and release actions contract and release the specified effector, and thus they have no arguments. The goto actions, on the other hand, move the agent's effectors about the blocks-world, and require an argument to specify the desired location of the end effector at the end of the movement. For example, the action LEFTGOTO(FRONT(a)) moves the left effector to the area in front of the block labeled "a." (See Section 5.2.3 for a description of the function FRONT.)

Each primitive action is defined by three sets of data: *possible results*, *result rules* and *relevant relations*. A complete definition the LEFTGRASP action is shown in Figure 5.4. *Possible results* are a set of likely outcomes of performing the action and provide a declarative statement about what the system believes might result from a given action. For example, a possible result of a LEFTGRASP is INLEFTHAND(X)

```
possible_result(leftgrasp, [leftgrasp, inlefthand(_)]).

result_rule(leftgrasp, [inlefthand(_)], [], []).
result_rule(leftgrasp, [lefthandempty, leftat(front(X)), leftnear(X)],
           [inlefthand(X)], [lefthandempty]).
result_rule(leftgrasp, [], [], []).

relevant_relations(leftgrasp, blocks_world, [on]).

possible_result(rightrelease, [righthandempty, rightrelease]).
```

Figure 5.4: ItPlanS definition for LEFTGRASP

for some object X. Notice that, the possible results of an action may not occur every time the action does. For example, INLEFTHAND(X) is true after a LEFTGRASP action if, and only if, LEFTAT(FRONT(X)) and LEFTNEAR(X) were true in the world before the action was performed.

Result rules describe the effects the system believes the action will have in given situations. As noted in Chapter 4, part of the situated reasoning used to decide on actions is simulating the performance of the action in the current world. Result rules provide the information needed to perform this simulation. Each of these rules is a conditional effect list with three parts. The first part, the *condition*, is a list of predicates which, must be true in the world for the rule to be applicable in simulating the action. The second part is a list of predicates that should be asserted to the simulation of the world model by the action, and the third is a list of predicates that should be removed from the world model as a result of the action. In the previous example, ItPlanS has a rule that states if LEFTAT(FRONT(X)) and LEFTNEAR(X) are true in the world model and a LEFTGRASP is simulated, then INLEFTHAND(X) will be true in the model. Likewise, ItPlanS has a rule which states if INLEFTHAND(X) is true and a LEFTGOTO(FRONT(Y)) is performed, then block "X" is no longer where it was.

Since result rules define what the system believes about the results of actions in the world, they may be in error. Result rules are not required to be physically accurate or that they be limited to those results listed in an action's possible result list. The only requirement is that they be complete and correct enough for the agent to achieve its ends. Optimally, for any action the union of the add lists of the result rules should subsume the possible results for the action, but this is not required. The distinction between the possible results and the result rules is maintained for exactly this possibility. If the possible results list of an action names a result not listed in one of the result rules, this tells us that the system believes some predicate can result from an action without knowing the conditions for its production.

```
possible_result(get(X), [inrighthand(X), inlefthand(X), get(X)]).
intention_mapping(get(X), [leftat(front(X)), maint, leftnear(X),maint,
        leftgrasp, ach], [inlefthand(X), get(X)], 1).
causes_failure(get(_), 1, [inlefthand(_)]).

intention_mapping(get(X), [rightat(front(X)), maint, rightnear(X), maint,
        rightgrasp, ach], [inrighthand(X), get(X)], 2).
causes_failure(get(_), 2, [inrighthand(_)]).
```

Figure 5.5: ItPlanS definition for GET

Since result rules look very much like STRIPS operators, I will explain how they are different. Result rules only give ItPlanS information about the results of performing an action in the world. Thus, all of the result rules could be replaced by a single "black box" that predicts the next state of the world given an action and the current world. Result rules only allow the agent to reason about the results of taking an action in a given world state. They do not prohibit the use of an action or constrain the times or world states that an action can be taken in. Thus, the conditions in result rules are not preconditions for the actions, and the result rules themselves are not STRIPS operators.

Finally, ItPlanS associates with each primitive action a set of *relevant relations* used in the process of simulating the effects of actions in the world. These relations tell ItPlanS which objects in the world must be included in an appropriate world model for simulating the action. In Figure 5.4 the relevance relations for the leftgrasp specify that, in the blocks-world, the "on" relation is relevant when constructing models for simulating leftgrasp actions. The use of these relevance relations will be covered in more detail in Section 5.3.2.

The objective of primitive actions is to provide a small set of actions from which larger behaviors can be built. The choice of actions as primitive is domain dependent, as some primitives may be helpful in one domain but not in another. It is assumed that, as an agent becomes more expert at a specific type of problem, they will learn new primitive actions to simplify the problem. Remember, primitive actions are the only actions that actually effect the state of the world; complex actions are realized as a series of primitive actions.

5.2.2 Complex Actions

As with primitive actions, ItPlanS definitions of complex actions have three parts: *possible results*, *intention mappings*, and *causes failure conditions.* A complete action definition for the complex action GET can be seen in Figure 5.5. The possible results lists of complex actions have all of the same problems and limitations as their primitive action counterparts. As with primitive actions they are only required to be a complete list of what the agent believes to be the common results of taking the complex action.

As mentioned before, ItPlanS views the results of a complex action as the sum of the results of the primitive actions chosen to satisfy its subintentions. Since the actions taken and the world context varies from occurrence to occurrence, there is no way to have complete beliefs about the results of complex actions. Thus, while complex actions have possible result lists, they do not have result rules.

Complex actions can be viewed as relations from a single intention to an ordered set of intentions. Rather than performing a physical action, complex actions map from an intention to a set of subintentions that need to be achieved, in order, for the original intention to be achieved. This relation between an intention and its subintentions is defined as the *intention mapping* relation. In ItPlanS, this mapping is not required to be one to one; therefore, there can be many expansions of any given complex action.

For example, in Figure 5.5, the action GET(X) can map to the list of intentions [LEFTAT(FRONT(X)),maint, LEFTNEAR(X),maint, LEFTGRASP,ach]. Remember each of these is an intention, not an action. The first two are intentions to act to achieve states, as discussed in Chapter 3. Also notice that each intention is paired with an intention types (maint or ach). If this set of intentions is achieved, in order, a stack action will result. However, GET(X) can also be expanded as [RIGHTAT(FRONT(X)),maint, RIGHTNEAR(X),maint, RIGHTGRASP,ach]. Notice that, the stack action's definition does not specify how to achieve any of these subintentions; any way to achieve them will be equally valid.

There is a problem in allowing multiple decompositions for a given complex action. Different decompositions may have different possible results. Therefore, the intention mapping relation maps a complex action to an intentional decomposition and a more specific set of possible results. For example, GET(X) is used to achieve intentions such as INLEFTHAND and INRIGHTHAND. However, each of the decompositions only achieves one of these possible states. Therefore, it is important that the intention mapping return the more specific expected results so that ItPlanS can choose the correct expansion. It would be irrational of the agent to attempt to satisfy the intention of having a block in its right hand by picking up the block with its left hand.

The third piece of information used to define a complex action is a set of possible failure conditions. These are defined in the system as *causes failure* conditions. For each decomposition of each complex action, there is a set of possible causes of failure that is only examined if action has failed. Causes failure conditions are part of the information ItPlanS needs to perform action repairs as discussed in Chapter 4. Section 5.3.4 will discuss action repair in ItPlanS and the use of these conditions in more detail.

At this point, I will repeat that none of the actions has any form of preconditions. Any action can be undertaken at any time and at any condition of the world. Of course, it will often be the case that, the action will not have the expected outcome or will fail if taken randomly; however, this does not preclude the agent from attempting to performing the action.

5.2.3 Specifying Locations

In order to provide a uniform treatment of block stacking to arbitrary heights in the blocks-world, it is necessary to represent space. To this end, I have defined a number of functions from objects in the world to areas of space around them. This treatment is not intended to be complete or rigorous, nor is it a substantive part of this work. It is a tool to solve some of the problems of designating locations in the blocks-world domain and nothing more.

The functions I defined are suggested by the English prepositions LEFT, RIGHT, FRONT, BACK, OVER, UNDER. These functions map from the argument/object to the spatial area described. For example, the action RIGHTGOTO(LEFT(b)) would move the right end effector to some arbitrary location to the left of block "b."

5.3 System Algorithm

ItPlanS's basic system algorithm is designed to find the next action to be executed. In this respect, it is very similar to NASL [21]. However, since ItPlanS is planning in the agent's space of intentions, the algorithm is a two-step process. First, ItPlanS finds its next unsatisfied intention. Then it decides on an appropriate action, by expanding the intention until a primitive action is found that can be executed. The following pseudo-code may help the reader get an overview of the system algorithm.

```
WHILE NOT(DONE)
  TRAVERSE(pos-intentions, next); sets NEXT to the next unsatisfied
                               ;  positive intention.
  EXPAND(pos-intentions, next,action); expands NEXT until it can be
                                ; satisfied by a primitive ACTION
  SIMULATE(action,conflict); simulate action, noting conflicts
  IF NOT(conflict); no conflict with the negative intentions
      THEN EXECUTE(action);
      ELSE RESOLVE(conflict);
END WHILE
```

Note that conflict resolution will lead to further intention expansion. Thus, each time the body of the WHILE loop executes, an action is produced. The following sections will discuss the algorithm in detail and explain how the system deals with failure.

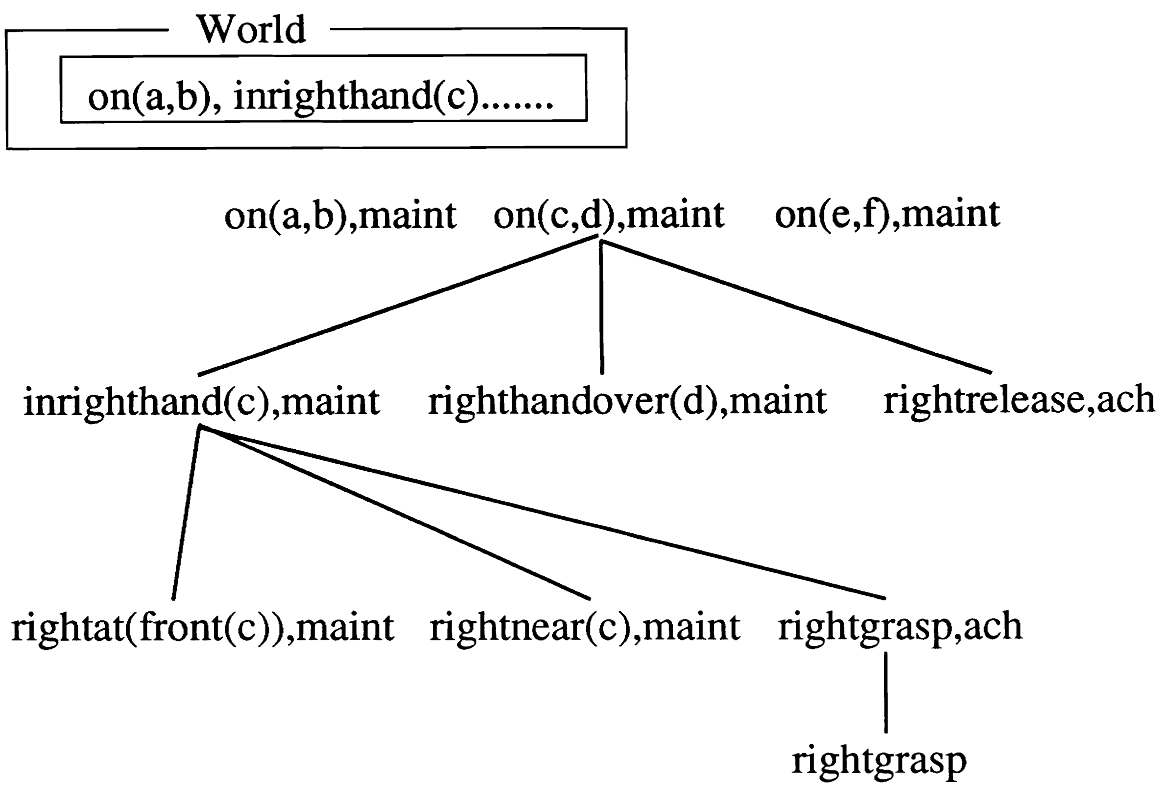

Figure 5.6: Intention structure before traversal and pruning

5.3.1 Intention Traversal

The structure used for representing the agent's intentions can be thought of as a forest of intention trees. Therefore the process of finding the first unsatisfied intention is a left to right, pre-order traversal of these intention trees. This traversal process also includes pruning satisfied intentions from the structure. This might best be seen in an example. Consider Figure 5.6. The agent's highest level intentions are the top three intentions shown: ON(a,b), ON(c,d), and ON(e,f).

Notice that, ON(a,b) and INRIGHTHAND(c) are already true in the world. As the agent begins to traverse its intentions from left to right, it first checks the world to see if ON(a,b) is true in the world. Since it is both true and a maintained intention, and its children have been pruned off, no action is taken and the search continues to the next intention. The agent now considers the intention ON(c,d). By checking the world, the agent realizes that it is not true. Since an expansion of this intention exists in the structure, the traversal process descends into this expansion and begins the same left to right traversal of its children. Checking the intention INRIGHTHAND(c), the agent verifies that this is true in the world. Since INRIGHTHAND(c) is a maintained intention and has been achieved, it will be left in the structure, but its decomposition will be deleted. This results in an intention structure shown in Figure 5.7.

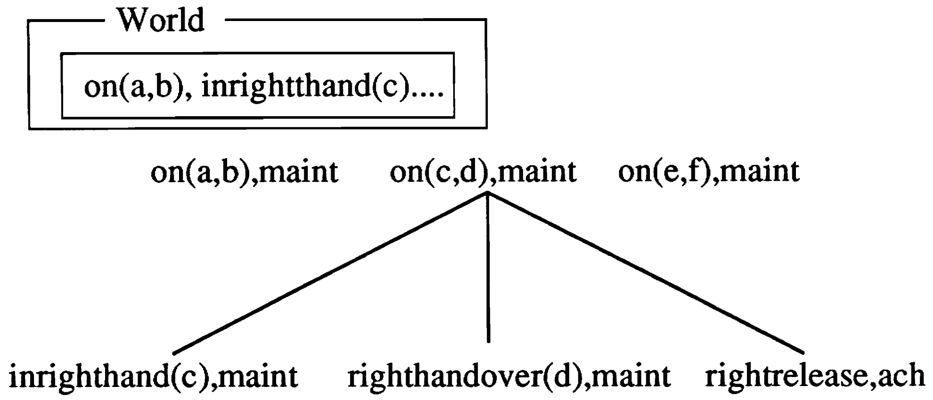

Figure 5.7: Intention structure after traversal and pruning

Having pruned the children of INRIGHTHAND(c), the agent next examines the intention RIGHTHANDOVER(d) and notices that this intention is not satisfied in the current world. Since this intention has no expansion in the structure, the traversal process ends giving RIGHTHANDOVER(d) to the intention expansion process.[1]

Notice that in looking for the first unsatisfied intention at each level of decomposition, ItPlanS is constantly reacting to the actual state of the world and to what it remembers performing, so that if an accident has occurred and the last action performed did not have the consequences the agent expected or if an earlier maintained intention has been invalidated, the agent can correct it immediately.

5.3.2 Intention Expansion

Having found the next positive intention to be satisfied by the agent, the algorithm enters the expansion phase. First, the algorithm searches the possible result lists of the primitive actions for an action that will satisfy the given intention.

Primitive Actions

If a primitive action will satisfy its intention, the system creates a minimal model of the world in which to simulate the action. This process uses the agent's *result rules* and knowledge of *relevant relations* in the following manner; first, the result rule, that will be applied in the simulation is selected. The objects listed in the condition of the rule are designated as "core" objects in the model, and the transitive closure of

[1]The complexity of the traversal portion of the algorithm is reassuringly small. The forest of intention trees can be made into is a single tree by creating an unsatisfiable root intention and making the system's intentions children of this root. The traversal algorithm's complexity is then only $O(k \log_k n)$, where k is the branching factor of the expansions and n is the number of nodes in a complete expansion of all of the intention trees.

the relevant relations over these core objects is taken to derive the salient objects for the model.

For example, suppose while planning in a blocks-world, the agent wanted to move a block that was the base of a four block tower. The selected result rule would place in the set of core objects the block to be moved and the block on top of it. However, this would leave out the two other blocks in the tower. Since the "on" relation is contained in the relevance relations for the move action, the other two blocks would be placed in the model for this action.

Having created the model, the result rule is then applied. By adding and deleting propositions as appropriate, a small model of part of the world after the action is taken is produced. If the resulting world model does not violate of any of the negative intentions of the agent, the action is considered acceptable and is carried out. Notice that, since the result rules are not required to be complete, this simulation may not produce all of the results that taking the action in the real world will produce, but this is to be expected. Human agents often do not foresee all of the results of their actions. Also, notice that a new model is created from the actual world state for each action. Therefore, there is no model maintenance cost for the planner, and there is no difficulty in keeping the model consistent with the actual state of the world.

Complex Actions

If no primitive action can be found that will satisfy the next intention, ItPlanS considers complex actions. Searching the possible result lists of each action, ItPlanS attempts to find a complex action that will satisfy its intention. If there are multiple complex actions that will suffice (or even multiple expansions of a single action) ItPlanS must decide among them. There are a number of factors to consider: ease of action, number of already achieved subintentions, the amount of reasoning required, and agent preferences. Thus, in general, the question of determining which expansion to use for a given intention is not easily answered. To make these decisions, ItPlanS has at its disposal, its immediate sensor data, its current positive intention structure, its negative intentions, all of its knowledge about actions, and the complete history of its actions to this point. Thus, it could use very complex selection methods requiring some, or all, of these resources.

However, the goal of this work is to give intentions their proper role in planning. Therefore, I have only implemented a few of the possible decision procedures that could be used to select an action expansion. Specifically, I have implemented some of the decision procedures that focus on the use of the intentional structure and the agent's knowledge of actions. Currently, ItPlanS first checks to see if there is an expansion for the action for which some of the intentions are already satisfied. If possible, this expansion is chosen since it will require less work to achieve.

The system also considers the possibility that a given expansion for the current intention may have an unintended effect that achieves part of a decomposition of the next intention. For example, suppose a block-crushing action is added to the blocks-world agent's repertoire, and this action can only be done with the right hand. Further suppose that the agent is left-handed, preferring to pick blocks up with its left hand. Now, if the agent has the intention to pick up a block, it will perform that task with its left hand. If the agent has the intention of crushing a block, it will pick the block up in its right hand and crush it. Now suppose, the agent has two separate intentions, first to get a block and then to crush that block. The agent should, knowing that it will crush the block, choose to satisfy the get intention with its right hand rather than its left. This will simplify the total set. This is exactly what ItPlanS does.

The system finally considers the possible overlap of the achievement of this intention with the next intention to be achieved. That is to say, the system looks at the next intention to be achieved and decides if there is a possible expansion of this next intention such that the expansion of the current action and the expansion of the next action both include the same subintention. Pollack [24] has referred to this kind of overlapping of intentional achievement as *overloading* intentions. The difference between this case and the previous case can be seen in the following way. In the previous case, the achievement of the first expansion has as an unintended effect, namely the achievement of one of the intentions of the second. In intention overloading, the expansions actually share a common intention; that is, the same intention occurs in both expansions.

In either case, ItPlanS not only chooses the complex action and expansion for the current action, but since the benefits to the system only result if a commitment is made to the breakdown of the next intention as well, it expands that intention as well. In this process, it makes a commitment to behaving in a certain way in the future. Thus, these kinds of choices develop the agent's future intentions.

If none of these optimizations are possible, the system must find an action and an expansion that merely satisfies the single intention in question and commit to this action/expansion pair. Having decided on an action, an expansion for that action, and possibly the next action as well, the actions and expansions are added to the positive intention structure. The system then traverses the expansion to retrieve the next intention to be achieved and expands this new intention. This process continues until a single primitive action is decided on. This primitive action is carried out; the system returns to the top most level of the intention structure and begins the search for the next action.

5.3.3 Conflict Resolution

There are a number of outstanding issues left out of this algorithm. Foremost is, what does the system do if there is no primitive action that does not violate one of the system's negative intentions? ItPlanS has a recovery mechanism for these situations which relies heavily on the accuracy of the action's result rules. What has happened in these cases is all of the examined actions have been determined to cause problems. The system needs to decide on the action it would "like" to take and then remove the causes of the problematic result of the action.

For example, suppose block "b" is stacked on block "a" and further suppose the agent is grasping block "a" in its right hand, but has not yet moved it. Let us also assume that the agent has an intention to move block "a" to a new position, and has a negative intention against breaking things. The only action the agent has available that will reposition block "a" is the action RIGHTGOTO. Finally, let us suppose the agent knows that if it performs a RIGHTGOTO action, block "b" will fall off and break. At this point, the agent is stuck. The only action to achieve its desired ends will violate a negative intention. Specifically, moving block "a" will cause a state where block "b" is broken, violating the agent's negative intention about breaking things.

Obviously, in this case, what the agent should do is determine the cause of the conflict (the presence of block "b" on block "a") and eliminate it (unstack block "b" from block "a"). This is exactly the process ItPlanS follows. If there is more than one possible action that can achieve the given intention, ItPlanS must first decide on which primitive action it will attempt to remove impediments to.

Having decided on an action, ItPlanS looks up the result rule that was applied in the simulation to produce the conflict. Its goal in this process is to prevent this result rule from being applied to the model of the world. This result rule was chosen because all of the predicates in the rule's condition were true in the world. If one of the predicates in the condition of the result rule were false, the rule would not be used to simulate the action and the conflict would not arise.

Let us return to our example to see how this works. The result rule that would be applied for this use of RIGHTGOTO would be something of the form [[INRIGHTHAND(X), ON(Y,X)], [ON(Y,table), BROKEN(Y)], [ON(Y,X), ON(X,Z)]] [2], where the first element is the condition, the second is the add list, the third is the delete list, "X,Y,Z" are all variables, and "table" is a constant. In order for this rule to be applied, it must be the case in the world that INRIGHTHAND(X) and ON(Y,X) are both true. In our specific example, INRIGHTHAND(a) and ON(b,a) are true so the rule is applicable. In order to keep this rule from being applied to the simulation, one of the elements of the conditional must be violated. That is, either INRIGHTHAND(X) or ON(Y,X) must be made false in the world.

At this point, ItPlanS chooses one of these conditions to invalidate, inverts the condition and achieves this inverted proposition as a positive intention. This will remove one of the conditions from the world making the rule invalid. Having done this, the original goto action can now be carried out without the undesired consequence of BROKEN(b). Of course, this method requires the agent's knowledge base to contain information about how to invert actions. In our example, the agent must know that

[2]In ItPlanS the variable Z is allowed to remain unbound, so that it may unify with any object that X might be on. This allows the system to delete all statements that X is on some object.

at least one way to undo ON(b,a) is to achieve ON(b,table).

ItPlanS must also decide on which condition to invalidate. It would be unfortunate if the system chose to invert one of the intentions it just achieved when there was another possibility. As a partial solution to this problem, the system maintains a list of those intentions that are "above and to the left" of the current one. That is to say, the system keeps a list of all the intentions it has tested in the traversal and expansion process which have been determined to be satisfied. It will obviously be preferable to invalidate an intention that is not on this list, since they are necessary for the achievement of the system's intentions and have been achieved already. Therefore, ItPlanS makes every effort to "protect" these intentions.

If it is impossible to find an intention that is not on the list of protected intentions then one from that list must be selected and inverted. This is the worst possible case as this intention must be marked as special to prevent the system from thrashing and undoing the work already done. This method allows for the temporary violation of maintained intentions.

Notice that ItPlanS utilizes the kinds of situated reasoning discussed in Section 4.2. The system does forward reasoning about the effects of primitive actions in specific situations. As just described, it also performs situated reasoning in order to eliminate conflicts. The next section will describe a set of limited reasoning processes that the program uses in correcting action failures. However, one of the primary concerns of the system has been the computational cost of reasoning. Therefore, the system, as it stands now, does the minimal amount of reasoning that it must in order to solve its problems. This leads the system to find less than optimal solutions for many kinds of problems. For example, right now ItPlanS will attempt to pick up objects even if its hands are full. Of course, once ItPlanS realizes the action will be impossible, it empties its hands and performs the action.

This inefficiency is not a result of incorrectness in the algorithm but of a failure to anticipate future problems. This of course can be solved by more reasoning. It is possible for ItPlanS to anticipate its inability to pick up the object by considering the contents of its hands and the object to be picked up. In fact, ItPlanS can use the same planning algorithm on a model of the world to make these predictions about action and plan interactions in the future. However, ItPlanS at this point does not perform this kind of advanced forward reasoning to preempt future conflicts. It only considers conflicts when they present themselves.

5.3.4 Action Failure and Recovery

As pointed out above, it is possible for actions to fail; primitive actions may not satisfy the intention they were invoked to achieve, or satisfying all of the subintentions in a complex action decomposition may not satisfy the original intention. With this in mind, ItPlanS has failure recovery mechanisms that follow the action repair paradigm outlined in Section 4.3.2 rather than re-planning. This is an issue of simplicity rather than a theoretically validated choice.

Primitive Actions

Unfortunately, the reasons that primitive actions fail, more often than not, have to do with the limits on performance of the agent's end effectors. For example, if an agent picks something up, it may slide out of the agent's hand, not because the agent picked the object up incorrectly, but rather, because the object was slippery. Another example is fast moving objects; if an agent tries to pick one up, the action may fail, simply because the object was moving too fast.

The only real remedy for this problem is to attempt the action again. If the agent tries to move to a specified location and doesn't get there because its treads have slipped, the agent must try again. This is exactly what ItPlanS does. Of course after a few attempts the agent should give up and attempt another method, but for now ItPlanS just continues to try the same action.

Complex Actions

Complex actions can fail for a number of reasons, some of which are obvious: for example, it is hard to pick up an irregularly shaped object if you already have something in your hands. Some reasons are less than obvious, such as McCarthy's potato in the tail pipe preventing a car from starting.

As said in Chapter 4, what all of these problems have in common is that unless the agent is aware that some condition may be a problem, there is nothing it can do about it. There is no general purpose recovery mechanism that is independent of having information about the possible causes of problems. ItPlanS follows this line of reasoning. As mentioned, each expansion of each complex action has a list of possible causes of failure, which if they hold in the world, may prevent the complex action from achieving some or all of its effects.

ItPlanS identifies complex action failure by checking to see if an intention's children have all been satisfied. If they have and the original intention is still unsatisfied, then the action has failed. The system then consults the list of possible causes of failure for this action and expansion. If a condition is found that is true in the world, this condition is undone. Afterward the complex action is attempted again, hopefully achieving the intention.

Chapter 6

Comparisons to Existing Systems

Another way to understand ItPlanS is in terms of how it differs from other planning systems. This will also help the reader clarify the issues that ItPlanS addresses and serve to establish where it would fit in a taxonomy of planning systems. To these ends, the following section is organized around existing systems and how ItPlanS differs from each.

6.1 Foundations

Much has been written about the failures of the early work in planning. However, since these systems are well known and well understood, it is instructive to compare new systems to them. Specifically, since these early systems confronted many of the most challenging problems in the field, any new solutions to these problems should be compared to them in order to better understand the new proposal. Therefore, this chapter will begin by considering two of the earliest planning systems.

STRIPS and PLANEX

I assume that the reader is familiar with the basic functioning of the STRIPS planner. From Chapters 2 and 5 many of the differences between ItPlanS and STRIPS should already be obvious: the lack of intentions in STRIPS, STRIPS's use of preconditions, STRIPS's use of a complete world model, and the hierarchical nature of ItPlanS.

Because STRIPS was the first system to confront the issue of plan execution, it is important to contrast the ability of STRIPS to accommodate changes in the world with ItPlanS ability to perform this task. In STRIPS, the functions of plan formation and plan execution are performed by two separate programs, STRIPS and PLANEX. In their later work on these systems, Fikes and Nilsson [9] introduced "triangle-tables" as a representation for plans that would allow the executor flexibility in executing plans.

Triangle-tables augmented STRIPS plans to include a table of all of the intermediate states of the world that result during a successful execution of the plan. Using this table, the plan executor could then recover from some failures and utilize unforeseen changes in the world state. By starting at the final state of the plan and traveling backwards through the derived plan states, the executor searched until a intermediate plan state was found that corresponded to the current world. Having found a state in the plan, the executor then knew which action to take.

The intermediate states generated in this process were also used by the executor to derive "macro" operators for subsequences of the actions in the plan. This was performed by selecting a goal and regressing through an existing plan, saving all of the steps required for generating the selected goal. This process also eliminated unneeded plan steps. This creation of macro operators allowed the plan executor to reuse parts of plans in new situations. Thus, the plan executor for STRIPS had two forms of flexibility in dealing with the world. First, it could repeat any specific subsequence of actions in a plan, if the plan had failed. Second, it could reuse subsequences of existing plans to achieve similar goals in the future.

The division of labor into separate planning and an execution monitoring programs in the STRIPS system stands in contrast to the integrated planning and immediate execution methodology of ItPlanS. As we have seen, ItPlanS calculates the next action to be taken, possibly making commitments about the future, and then executes this single action. There is a whole family of planners that integrate planning and action in this way [11, 13, 21], to avoid the problems associated with separating planning from action, and enable the systems to be more responsive. Since the system can begin to act before a complete plan has been derived, the its response time is decreased. While this leaves the system open to deriving less than optimal plans, this is an acceptable loss for increased responsiveness.

Stepping beyond the problem of separating planning and action, there are limitations associated with the use of triangle tables in terms of reactivity and deriving macro operators. First, triangle-tables only provide limited improvement in the system's ability to cope with failure. The intermediate states derived in the triangle-tables are the intermediate states in a successful execution of the plan. Thus, this method will not be successful if a failed action causes some change in the world that is not a normal stage in the plan.

The derivation of macro operations in STRIPS also has a problem. The algorithm for creating a macro operator calls for collecting all of the actions that are part of the "proof" of the action. However, given that this "proof" was used in the context of another goal, it may in fact be an inefficient way to achieve the goal in a new context. For example, suppose that the series of three actions $[\alpha, \beta, \gamma]$ derive two conditions, A and B. Now, also suppose that action ω also achieves A but does not achieve B. If in another context, we again wish to derive A, the system macro for this goal could be $[\alpha, \beta, \gamma]$ rather than ω. This would be inefficient, and even inappropriate if the agent explicitly did not want B to hold. In this case, the only recourse is to leave the plan executor and plan from first principles to achieve the goal of A. This limits the applicability of these macros to domains with relatively independent goals.

The method used by ItPlanS solves all of these problems. ItPlanS gains responsiveness by integrating planning and action, and does not fall prey to STRIPS's problems in recovering from failure because the planner replans for the achievement of this goal as it would for any other system goal using the world state, memory, and situated reasoning. Thus, ItPlanS avoids both of the errors found in triangle tables; since it uses the actual state of the world in re-planning, ItPlanS is not limited to only considering the intermediate states of the correct plan but can react to the actual state of the world and the changes caused by the failure. Moreover, since planning is always done from the world state in the context of the current intentions, ItPlanS is not limited to using macros developed for other sets of goals, but can rather, tailor its work specifically to the setting encountered.

NOAH

Sacerdoti's [26] systems, NOAH and ABSTRIPS were the first hierarchical planners. In ABSTRIPS, the hierarchical nature of the planning was accomplished through the priority of achieving various preconditions. This stands in contrast to his later work on NOAH which has explicit action expansion similar to the process used in ItPlanS.

NOAH suffers from many of the same problems as STRIPS. For example, NOAH uses preconditions to encode situation-dependent information. NOAH, like STRIPS, also assumes that the world is predictable and that actions have "defined" results. That is, every action has a fixed and finite set of possible reasons for its use.

However, NOAH does possess an interesting ability that ItPlanS lacks: NOAH is a nonlinear planner. Thus, the ordering for the achievement of the system's goals is determined by interactions between the goals, and ordering commitments are only made when required by some constraint in the plan. In ItPlanS, I have chosen not to consider the issue of nonlinear planning. I believe the ordering of goals based on their interactions is an issue of learning the interactions that are common in a given situation and how to select action expansions with the correct ordering for the situation. In ItPlanS, intentions are ordered, and this ordering is assumed to be relevant. This means that the two sequences of intentions, [goto(store), buy(food)] and [buy(food),goto(store)] can produce very different behaviors. In general intentions to act in the future are fixed in this manner. Thus, it is not possible to arbitrarily reorder them in order to perform plan optimization.

6.2 Reactive Systems

In response to the need for agents to interact in environments with significant time constraints, a number of researchers have proposed theories of agent-environment interaction that have been called reactive or reaction systems. These systems stress the need for an agent to be able to respond to the world as rapidly as possible.

Pengi, Sonja, and Rex

Agre, Chapman, Kaelbling, and others [1, 2, 5, 19] have argued that the most important ability an agent can have is the ability to respond rapidly to its environment. To this end, they are willing sacrifice actual planning. Essentially they encode the desired responses of the agent as a function from sensor inputs to effector outputs.

The results of this are straightforward. First, none of these systems actually engage in activities that could be called planning. While they do engage in planned activities, all of the planning has been done by the system designer before construction. This approach also makes certain commitments to the nature of the agents. Since these agents are doing nothing more than calculating a fixed function, they conform to what might be called a behaviorist school of agent action, in which an action is nothing more than a response to the stimulus provided by the world.

Of course, these systems can produce arbitrarily complex behavior and do so at high speeds. However, the fact that a given Turing computable function can be encoded in hardware is well known, and since any planning algorithm could be expanded into hardware, in this way, systems can be designed which have the same behavior as any planning system. However, this does not contribute in a meaningful way to understanding how agents actually plan. In their favor, these systems are the only ones, discussed here, that do not have explicit preconditions in them; however, this is hardly surprising considering their nature.

Universal Plans

Marcel Schoppers [27], has suggested another way of achieving fast reaction times in agents. Briefly, his proposal is to consider off-line all of the possible world states and to construct plans that lead to the goal from each of these different states. The agent then uses the actual world state to index into these *universal plans* to determine the action which will carry the agent toward its goal. This allows the agent to constantly select the correct action to move toward its goal, no matter what might happen in the actual world.

Unfortunately, this method, while achieving the goals of fast reaction time, winds up having some of the problems of both the STRIPS approach and the reactive work mentioned before. Since creating the universal plans happens off-line, these plans suffer from some of the problems of disassociating planning from action. For example, universal plans are limited by the accuracy of the world model used to perform the planning. If the model employed during this phase is not complete with respect to all possible states of the world, the generated plans will not be universal. Since the "off-line" planning process must traverse all of the possible world states and must include an entry for these states, this process is unrealistic for any but the simplest domains and tasks. Schoppers, in his thesis, admits that "sufficiently large domains are simply not solvable by the current plan synthesis machinery."[27, page 115] Finally, once a universal plan is built it has all of the rigidity and domain dependency that the reactive systems have. In some sense, these universal plans are the functions Agre and Kaelbling build their systems to calculate.

Schoppers, however, is the only planner that I will examine that makes a distinction between actions and effects. This difference is embodied in ItPlanS in result rules. The use of result rules to determine the effects of an action in a given world state is similar to reasoning done by Schoppers's universal plans.

6.3 Integrated Planning and Action

We can see from these last two systems, the attempt to increase system reactivity at the cost of planning can pay off, but only at the cost of generality. The next section examines a series of planning systems where the functions of planning and acting are interleaved, as in ItPlanS; therefore, I will attempt to emphasize the ways in which ItPlanS is an improvement over these systems.

NASL

NASL [21] was the first planner to interleave planning and execution. The most striking similarities between ItPlanS and NASL are the hierarchical nature of the planning process and the fact that complete plans are not devised before execution begins.

ItPlanS goes beyond NASL to solve two of the limitations McDermott lists at the end of his paper. The first limitation is that NASL lacks any concept of "success conditions" for actions. In ItPlanS, the success conditions of actions are derived from the explicit representation of intentions. The success or failure of an action is determined solely by the achievement of the intention that invoked the action.

McDermott also lists as one of NASL's limitations, that it has no theory of error correction. As we have seen, ItPlanS does have such a theory. While this theory is knowledge dependent, it carries ItPlanS much further than NASL. Moreover, NASL, unlike ItPlanS, is incapable of making commitments about future action. This limitation makes it more likely that the planner will find sub-optimal plans and make unrecoverable errors.

RAPs

The RAP system [11], was specifically built to examine decision making without considering future states. Therefore, it would be unfair to attack it by pointing out the benefits that intentions yield in looking forward to help make local decisions.

The RAP system architecture is similar to the ItPlanS architecture in many of the same ways as NASL. The RAP system is given an initial set of RAPs(reactive action packages) placed in a queue. A RAP can be thought of as "... an autonomous process that pursues a planning goal until that goal has been achieved." [11] When a RAP reaches the head of the queue, the world model is examined to determine if the RAP's goal has been achieved. If it has, the RAP is removed from the queue, and the next RAP is considered. If the goal is not true in the world model, then the RAP is examined further. If the RAP is a "primitive command", it is sent off for execution. If not, then it is hierarchically expanded into subRAPs which are placed on the queue.

Beside the issues of local decision making and a lack of future commitments, ItPlanS improves over the RAP system in at least three significant ways. First, the RAP system attempts to maintain a complete world model. Firby admits that it is possible "for the world model to become inconsistent with the state of the real world." [11, page 205]. Even thought the system knows that this inconsistency is possible it ignores this possibility and assumes that the world model is always correct. Thus, if the world model of the RAP system is in error, then an action may be issued that is inappropriate.

Firby discusses a case where the agent's incomplete knowledge causes the world model to be in error, a case of an agent attempting to lift a rock that is too heavy. Firby's solution depends on the assumption that the information the agent needs is not immediately available in the world. If rock were clearly affixed to the ground, then the action of lifting the rock should not be attempted. However, since this information might not be in the RAP system's world model, the action would still be attempted. Thus, there are cases where information in the environment should prevent the performance of actions, but since Firby's agents are planning from their world model they will fail in these situations.

ItPlanS avoids this problem by constantly appealing to its direct sensor input, rather than a model. This also allows it to avoid maintaining a complete world model that is accurate enough to capture all of its knowledge about the effects of actions and still have the information necessary to simulate actions.

Second, the RAP system is unable to cope with interactions among its RAPs; if an interaction between RAPs is discovered, the RAP whose goal has been interfered with is terminated and returns failure. ItPlanS, not only is capable of dealing with these kinds of interactions, but the use of intentions gives a principled account of how long to allow the violation of a subgoal.

Finally, ItPlanS makes a distinction between maintained intentions and achievement intentions that the RAP system does not make. This distinction means that when the ItPlanS algorithm terminates, all of its positive intentions of have been achieved, and all of its maintained state intentions hold in the world. RAPs on the other hand can be fooled by goal interactions. If a top level RAP succeeds and is removed from the process queue, then a subsequent RAP may undo the system's previous effort. This cannot happen in ItPlanS.

PRS

Of all of the systems examined here, the system most similar to ItPlanS is the PRS by Georgeff and Lansky and Ingrand [12, 13]. ItPlanS and PRS are so similar that rather than a discussion of the PRS algorithm, this subsection will focus on some more substantive theoretical similarities and differences between the two.

As with the most of the systems discussed, PRS employs preconditions in its action representation. I have already noted the limits this places on the use of intentions. PRS, unlike the other systems, does represent the intentions of the agent. However, there are significant problems with PRS's treatment of them. To begin with, while PRS does have an explicit representation of positive intentions and it can perform situated reasoning using them, it ignores negative intentions. Thus, PRS is forced to encode them within the preconditions and expansions of actions.

There is a second problem with the use of intentions in PRS. In ItPlanS, the expansion of an intention can exert an effect on those intentions that follow it. As we have seen, the agent can make a commitment to a method of expanding a future action on the basis of an interaction between the current intention and a future one. PRS does not have this ability. In PRS, intentions are only the methods or plans for achieving a goal, and are only used to prevent PRS from constantly re-planning new methods of achieving goals. This interpretation ignores the role intentions can play in constraining possible methods of satisfying the agent's goals.

PRS does have an ability that ItPlanS lacks. Like NONLIN, PRS is able to change the order of evaluation of its intentions. In ItPlanS, the task of ordering intentions is considered a problem beyond the planner.

6.4 Other Systems

Finally, there are two systems that should be mentioned that do not seem to easily fall into one of the categories discussed so far. In an effort to be complete they have been included in their own section.

Vere and Bickmore

Vere and Bickmore [31] describe a "complete integrated agent" called Homer, a mobile, autonomous submersible that performs activities in a simulated area of ocean in and around a harbor. Homer displays unusual traits, that include maintaining positive and negative intentions.

Homer can be instructed to perform an action at a later date. This instruction will be translated into the goal of performing the action at the specified time. When the correct time arrives, Homer will dutifully perform the action. In the same way, Homer can be instructed not to perform an action on a given day, and on the given day he will not perform any action that would require his violating this directive. These examples would seem to indicate that Homer possesses the functional equivalent of positive and negative intentions.

Unfortunately, this process is performed by the use of temporal constraints on the planning process, rather than explicit intentions. That is to say, Homer's "intentional" behavior is achieved without explicit intentions. Thus, Homer will be unable to use his intentions in more powerful ways, for example jointly satisfying multiple goals without being specifically instructed to do so. I have included this system because of it displays the appearance of having both positive and negative intentions. To the best of my knowledge, it is the only system other than ItPlanS that appears to have both types of intentions.

Chef

The final system that will be examined here is the case-based planning method of Kristian Hammond [15]. The method he has proposed is radically different than other systems. In brief, he suggests that planning, is a memory task rather than a cognitive one. That is, when confronted with a planning problem, an agent retrieves from its memory a complete plan for a similar task based on key features of the problem. The agent then attempts to anticipate possible problems in applying this plan to the current situation. If the agent anticipates a problem, a solution is found and the changes are made to the plan before execution. The plan is then executed and the results observed.

If the plan fails on execution, the plan will be "fixed" and stored in the agent's knowledge base under an index of the goals it satisfies and the failures it avoids. It is important to remember that the agent is storing complete plans for activities in its knowledge base, and then when confronted with a need to plan it retrieves from its memory the whole of this "closest" plan.

This algorithm has at least two significant problems. First, it forces the agent to commit to a whole of plan at the level of primitive actions in a single step. As Hammond writes,

> The first component of a case-based planner's memory of plans is the plans themselves. These plans are represented as a fully ordered set of steps at the level of the planner's primitive actions. [15, page 72]

This means, when one of these plans is pulled from the memory the agent retrieves and commits to the whole plan at one time. For example, if an agent wanted to go to Europe and the closest experience it had was a bus trip to New York. This bus plan would be retrieved and the agent would have immediate access to all of its memories about walking through the bus station as well as all of the primitive actions associated with that plan to get to New York. This is unrealistic.

Second, Hammond's algorithm must specify what it means for an existing plan to be the "closest" match. Hammond acknowledges that the question of what defines the best match depend of the goals and the initial world state, but does not go further. Obviously this information must be tempered by other situational information. For example, the systems other intentions, both positive and negative, and available resources (including time).

With this criticism aside, Hammond's work does have a kernel of truth. Memory obviously plays an important role in providing possible plan expansions for a given intention. This kind of indexing of possible expansions based on previous information has not been used in ItPlanS precisely because it opens the questions of determining similarity and closeness discussed above. However, memory and some kind of closeness measure must eventually play a role in filtering out inappropriate plan expansions in a system like ItPlanS.

Chapter 7

Conclusions

Chapters 2 of this paper presented the problem that preconditions pose for intentions and their role in means-end reasoning in planning. Chapter 4 then went on to propose a solution to this problem, by replacing preconditions with explicit representations of intentions, situated reasoning and robust failure mechanisms. In Chapter 5, I described the ItPlanS system as a functioning example of these ideas. Thus, I have defined the problem and proposed a solution. However, I have not yet proved the validity of this solution method. Thus, I propose to focus primarily on quantifying the scope and abilities of a system like ItPlanS. This chapter will detail some specific proposals for further work.

7.1 Waiting as an Intentional Activity

In examining work on planning continuous processes, I realized that (with a few notable exceptions [31, 33]) waiting, as an action, has been left out of most discussions of planning. I believe that an agent's intentions provide the information needed to determine when it is appropriate for the agent to wait, and how long to wait for. Thus, I have begun to look at waiting as an intentional action, starting with one of the traditional problems in this area: attempting to fill a container with a fluid from a tap. The agent must position the container under the tap, turn the tap on, wait for the container to fill, and then turn the tap off. The planner must make the crucial decision not to act for some period of time.

Many planners do not consider waiting to be a planned activity. Therefore, they must solve the waiting problem with ad-hoc solutions. I am proposing that an agent's intentions allow it to know when it is appropriate to wait. This work takes the position that any state of the world may be brought about by waiting. Since there are other agents in the world, rather than remaining the same, the world will usually change during any period of inaction. In fact, almost anything may happen if the agent waits long enough.[1] Obviously this is not true of volitional actions, but any state may come into being if the agent waits long enough for another agent to bring it about. In ItPlanS, this would be captured by the having a primitive action *wait* with a possible result condition that matches against any state intention.

The agent can therefore consider the wait action for achieving every goal. In other systems, this might lead to the agent waiting for everyone else to accomplish its goals for it. However, in ItPlanS, the action must successfully pass through a simulation step before the agent commits to executing it. To prevent our agent from complete inaction, only those conditions under which an agent should reasonably believe that a change will occur in the world will be included as result rules. Thus, only in the cases where the simulation of the action satisfies the agent's intention will the agent actually wait.

For example, if the agent were attempting to fill a bucket with water, the agent would have the following result rule **result_rule(wait, [under(tap, bucket), positive_flow(tap)], [full(bucket)], [])**. As before, the first element is the action, the second the condition, and the third and fourth are the add and delete lists for the rule. Thus, the agent believes that waiting, while a bucket is under a tap and the tap flowing, will result in a full bucket. Therefore, if the agent desires the bucket to be full and these conditions hold, the agent will wait.

I have done some preliminary work towards a principled, intention-based treatment of waiting. However, this theory has yet to be tested and experimentally verified. Therefore, I propose to incorporate this intentional treatment of waiting into ItPlanS.

7.2 Formalizing ItPlanS Performance

To date, ItPlanS has only been used in two domains; both of them have been simplified for the system's use, a simple blocks-world, described earlier, and the AnimNL project [32]. Unfortunately, while these simple domains can be looked on a proof of concept, they are not complex enough to allow me to answer certain obvious questions about the abilities of the system. For example:

- How well can ItPlanS avoid conflicts with its negative intentions using only a one step look-ahead?
- How well does the system recover when its negative intentions are compromised?
- How successful are the failure recovery mechanisms?
- Are there other obvious local intentional optimizations that will allow even better performance than those I have considered?

[1]As the saying goes "All things come to those that wait."

To answer these questions, I propose to move the ItPlanS system to a more complex domain. One possibility is to add more detail and actions to the two-handed blocks-world I have been using. These actions could include an operation to paint blocks that would involve waiting for the paint to dry and block shape-changing operations similar to the "crush" action described in Chapter 5. As with the crush action, these operations would have to be performed by a specific hand. I further propose to include more complex stacking structures to the system, like arches or pyramids, and to include in the model a random possibility for actions to fail.

I believe that with these additions I will be able to completely test the abilities of ItPlanS and answer the outstanding questions. Since the robot will still have two hands, almost all of the agent's possible intentions will have multiple ways of achieving them. Thus, this new model will provide more than enough complexity for the analysis of negative intention conflict. The inclusion of the random failure of actions will allow me to completely test the action failure mechanisms I have placed in the system, while the "painting" action has been included in order to test the theory I have put forward about waiting.

7.3 Formalizing ItPlanS Algorithm

Once the system has been moved to a more complex environment. I hope to be able to quantify exactly how often there are negative intention conflicts and what properties of the environment make these conflicts likely. I would also like to be able to describe the classes of problems that are hard for ItPlanS to solve, so that, domains where the ItPlanS can be used most effectively can be identified. Finally, I will attempt to provide a more formal specification of the system's algorithm. To this end, I have looked briefly at a very different problem that has the same underlying structure as ItPlanS algorithm, natural language generation by mildly context sensitive grammars.

Mild Context Sensitivity

Surprisingly, there are a number of similarities between the process ItPlanS carries out and the problem of natural language generation using mildly context sensitive grammars like Tree-Adjoining Grammars(TAGs) [18] and Combinatorial Categorial Grammars(CCGs) [28]. This similarity between planning and natural language generation is not an obvious one; therefore, I will try to explain the connection.

Borrowing terms from formal language theory, we can call a planner which only looks at its goal when making planning decision a *context free* planner, while a planner that looks arbitrarily far into the future and the past could be called *context sensitive.* However, ItPlanS does not have either of these properties; the system falls somewhere between context free and context sensitive. That is, when ItPlanS expands a complex action, among other things, it examines the next positive intention in its structure to inform its decision, but this is a bounded, one-step look ahead. Thus, ItPlanS considers more than its single goal but less than the whole of the possible plan.

Mildly context sensitive languages, like TAGs and CCGs, display a similar bounded sensitivity to the situation. I believe there are more parallels between the areas of planning and natural language generation using these grammatical formalisms. Since the theory underlying CCGs and TAGs is more formal and rigorous then the theory underlying ItPlanS, I hope these parallels will help in providing a more formal description for ItPlanS's operation and abilities.

7.4 Conclusions

In closing, I would like to briefly summarize the claims of this paper and state my proposals for future work. In this proposal I have argued that preconditions are always defined relative to an assumed set of goals. Therefore, they limit the application of intentions to the process of means-end reasoning required by the planning process and must be removed from action representation. I have further argued that, the roles preconditions have performed can be filled by the use of explicit representation of positive and negative intentions, situated reasoning, and robust failure mechanisms. I have presented the Intentional Planning System (ItPlanS) as functioning example of these ideas, and compared it to a number of existing planning systems.

I have gone on to propose to answer a number of questions about the abilities of the Intentional Planning System, specifically, the effectiveness of the one step look-ahead to avoid negative intentions and the effectiveness of the failure recovery algorithms. To this end I am proposing create a more complex environment for the ItPlanS system to interact with. The two-handed blocks-world that is described in this paper would be enhanced with new actions that require waiting for their completion, as well as actions with fixed resource requirements, and the possibility of primitive action failure.

Bibliography

[1] Philip Agre. The dynamic structure of everyday life. Technical Report 1085, MIT Artificial Intelligence laboratory, 1988.

[2] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Sixth National Conference on Artificial Intelligence*, 1987.

[3] James Allen. Temporal reasoning and planning. In *Reasoning about Plans.* Morgan Kaufman, 1991.

[4] Michael Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.

[5] David Chapman. *Vision, Instruction and Action.* MIT Press, 1991.

[6] Philip Cohen and Hector Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[7] Philip Cohen, Jerry Morgan, and Martha Pollack, editors. *Intentions in Communication.* MIT Press, 1990.

[8] Donald Davidson. *Essays on Actions and Events.* Oxford University Press, 1980.

[9] Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[10] Richard Fikes and Nils Nilsson. A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[11] R. James Firby. An investigation into reactive planning in complex domains. In *Sixth National Conference on Artificial Intelligence*, 1987.

[12] Michael Georgeff and Francois Felix Ingrand. Decision-making in an embedded reasoning system. In *International Joint Conference on Artificial Intelligence*, 1989.

[13] Michael Georgeff and Amy Lansky. Reactive reasoning and planning. In *Sixth National Conference on Artificial Intelligence*, 1987.

[14] Alvin Goldman. *A Theory of Human Action.* Princeton University Press, 1970.

[15] Kristian Hammond. *Case-Based Planning: Viewing planning as a memory task.* Academic Press, 1989.

[16] Steve Hanks. Practical temporal projection. In *Ninth National Conference on Artificial Intelligence*, 1990.

[17] Gilbert Harman. *Change in View.* MIT Press, 1986.

[18] Arvind K. Joshi and Yves Schabes. Tree-adjoining grammers and lexicalized grammers. In *Definability and Recognizability of Sets of Trees.* Elsevier, 1991.

[19] Leslie Pack Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, 1990.

[20] Vladimir Lifschitz. On the semantics of strips. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop.* Morgan Kaufman, 1987.

[21] Drew McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[22] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., 1980.

[23] Martha Pollack. *Infering Domain Plans in Question-Answering.* PhD thesis, University of Pennsylvania, 1986.

[24] Martha Pollack. Overloading intentions for efficient practical reasoning. *Nous*, 25:513–536, 1991.

[25] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[26] Earl Sacerdoti. The nonlinear nature of plans. In *International Joint Conference on Artificial Intelligence*, 1975.

[27] Marcel Schoppers. Universal plans of reactive robots in unpredictable environments. In *International Joint Conference on Artificial Intelligence*, 1987.

[28] Mark J. Steedman. Type-raising and directionality in combinatory grammer. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1991.

[29] Gerald Jay Sussman. *A Computer Model of Skill Acquisition.* Elsevier, 1975.

[30] Austin Tate. Generating project networks. In *International Joint Conference on Artificial Intelligence*, 1987.

[31] Steven Vere and Timothy Bickmore. A basic agent. *Computational Intelligence*, 6:41–61, 1990.

[32] Bonnie Webber, Norman Badler, F. Breckenridge Baldwin, Welton Becket, Barbara Di Eugenio, Christopher Geib, Moon Jung, Libby Levison, Michael Moore, and Michael White. Doing what you're told: Following task instructions in changing, but hospitable environments. Technical Report MS-CIS-92-74, University of Pennsylvania, 1992. Submitted to the *AI Journal*, Special Issue on Computational Theories of Interaction and Agency.

[33] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.