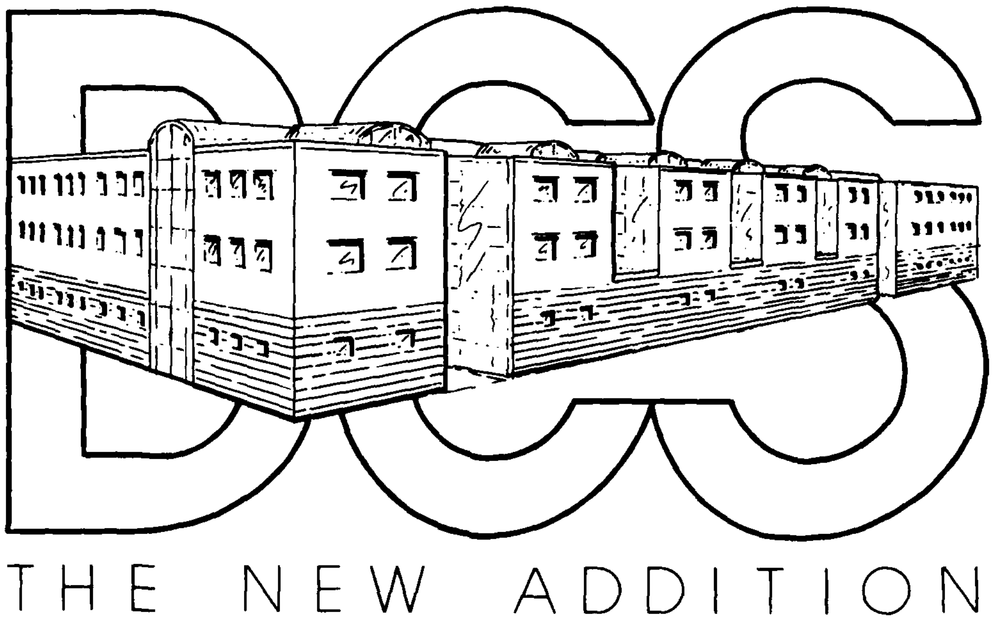
DTIC FILE COPY

(2)

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

AD-A223 434



REPORT NO. UIUCDCS-R-90-1605 UILU-ENG-90-1744

DTIC
S ELECTE JUN 12 1990 D
D

LEARNING COMPLETABLE REACTIVE PLANS THROUGH ACHIEVABILITY PROOFS

by

Melinda Tumaneng Gervasio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

May 1990

90 06 11 009

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UIUCDCS-R-90-1605	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Dept. of Computer Science University of Illinois; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 1304 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0309
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Learning Completable Reactive Plans Through Achievability Proofs

12. PERSONAL AUTHOR(S)
Melinda Tumaneng Gervasio

13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM ________ TO ______	14. DATE OF REPORT (Year, Month, Day) 1990 May	15. PAGE COUNT 81

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
05		10

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
classical planning, reactive planning, explanation-based learning, imperfect domain theory problem

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis presents an integrated approach to planning wherein a classical planner is augmented with the ability to defer achievable goals and address these deferred goals during execution. This integration gains from reactive planning the ability to utilize runtime information, thus reducing the need for perfect a priori information, while retaining the goal-directedness afforded by a priori planning. This approach also retains the provably-correct nature of plans constructed by a classical planner by requiring that all deferred goals have achievability proofs guaranteeing their eventual achievement. Proving achievability is shown to be possible for certain classes of problems without having to determine the actions to achieve the associated goals. General plans for use in this integrated approach are learned through a modified explanation-based learning strategy called contingent explanation-based learning. In contingent EBL, deferred goals are represented using conjectured variables, which act as placeholders for the eventual values of plan parameters whose values are unknown prior to execution. Completors are incorporated into general plans for the runtime determination of values to replace the conjectured variables. Since only conjectured variables with accompanying achievability proofs are allowed into contingent explanations, the general plans learned in contingent EBL are guaranteed to be completable. An implemented system demonstrates the use of contingent EBL in learning general completable reactive plans, which enables the construction of robust, efficient plans for spaceship acceleration.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Alan Meyrowitz	22b. TELEPHONE (Include Area Code) (202) 696-4302	22c. OFFICE SYMBOL ONR

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

LEARNING COMPLETABLE REACTIVE PLANS THROUGH ACHIEVABILITY PROOFS

BY

MELINDA TUMANENG GERVASIO

B.C.S., University of the Philippines, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana–Champaign, 1990

DTIC
COPY
INSPECTED

Urbana, Illinois

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification ....
By ....
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

LEARNING COMPLETABLE REACTIVE PLANS THROUGH ACHIEVABILITY PROOFS

Melinda Tumaneng Gervasio, M.S.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1990
G. F. DeJong, Advisor

This thesis presents an integrated approach to planning wherein a classical planner is augmented with the ability to defer achievable goals and address these deferred goals during execution. This integration gains from reactive planning the ability to utilize runtime information, thus reducing the need for perfect a priori information, while retaining the goal-directedness afforded by a priori planning. This approach also retains the provably-correct nature of plans constructed by a classical planner by requiring that all deferred goals have achievability proofs guaranteeing their eventual achievement. Proving achievability is shown to be possible for certain classes of problems without having to determine the actions to achieve the associated goals. General plans for use in this integrated approach are learned through a modified explanation-based learning strategy called *contingent* explanation-based learning. In contingent EBL, deferred goals are represented using conjectured variables, which act as placeholders for the eventual values of plan parameters whose values are unknown prior to execution. Completors are incorporated into general plans for the runtime determination of values to replace the conjectured variables. Since only conjectured variables with accompanying achievability proofs are allowed into contingent explanations, the general plans learned in contingent EBL are guaranteed to be completable. An implemented system demonstrates the use of contingent EBL in learning general completable reactive plans, which enables the construction of robust, efficient plans for spaceship acceleration.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Gerald DeJong, for providing much direction in this research. Through countless discussions with him, vague ideas were transformed into concrete issues and rough approaches refined. His intuitions about worthwhile endeavors have been invaluable in focusing this work.

It has been very rewarding to work within the creative, stimulating environment which past and present members of the research group—Michael Barbehenn, Scott Bennett, Alex Blanco, Steve Chien, Jon Gratch, Ray Mooney, Shankar Rajamoney, Jude Shavlik, and Larry Watanabe—have helped provide. Special thanks go to Steve Chien for providing the insight into nested implications, which finally enabled us to overcome a persistent representational problem in the construction of achievability proofs.

I would also like to thank Mark and Debs for patiently listening to the ramblings–on of an AI person about crossing the street and hammering nails, and providing fresh viewpoints on the matter.

Finally, I would like to thank my family for their unwavering love and support through the years. The steady encouragement from home has helped immeasurably in the pursuit of this degree.

This research was started at the Coordinated Science Laboratory of the University of Illinois at Urbana–Champaign and continued at the Beckman Institute for Advanced Science and Technology. The thesis and the implementation were written using equipment donated by the IBM Corporation. This work was supported by the Office of Naval Research under grant N00014–86–K–0309.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

It is mid–afternoon and your stomach growls for something more substantial than the stale, salty chips and the sugar–based cupcakes offered by the vending machines in the basement of your office building. The vision of two large scoops of ice–cream nestled in a bed of sweet strawberries and smothered with hot fudge and whipped cream from the shop across the street pops into your head. This suggestion, which quickly wins the approval of every executive module in your brain, is passed onto your planner, which takes virtually no time to formulate a plan to take you out of your office, into the elevator and down to the first floor, out of the building, across the street, and into the shop....

This is a simple problem and yet one which the majority of today's planners are ill–equipped to handle. Classical planners [Chapman87, Fikes71, Sacerdoti77, Stefik81, Sussman73] determine provably–correct plans completely prior to execution and thus require perfect a priori knowledge. In the sundae problem, this means knowing the height above the floor of the elevator "down" button so that the reaching action can be formulated, which cars will be on the road at the time of crossing so that the time of crossing can be determined, and the precise location of the ice–cream shop so that the path to take can be planned out. It also means correctly computing a priori the precise number of steps to walk from the desk to the door, the force to use to open the front door, and the speed at which to cross the street. Classical planners require complete and correct domain theories, which in the real world often means intractability. Reactive planners [Agre87, Firby87, Schoppers87, Suchman87], on the other hand, construct plans dynamically during execution and thus avoid most of the problems of inference by providing sensitivity to the runtime environment. Unfortunately, with their inability to look beyond the current state and reason about their actions in relation to their goals, reactive planners may make decisions which obstruct goal achievement. Thus, in the search for a sundae, preventing a collision with a speeding car while crossing the street—assuming that this point in the plan is reached in the first place—might be achieved by stopping in the middle of the street and causing the maniacal driver to swerve off the street, up the sidewalk, and into the ice–cream shop. This resulting sequence of events, however, might also result in blocking the entrance to the shop and the shop's subsequent closing, preventing the acquisition of a hot fudge sundae. Furthermore, reactive planners are highly problem–dependent and domain–dependent, complicating the use of learning strategies for extending problem–solving capabilities.

There thus seem to be several competing requirements for the ideal planner. It must be able to guarantee goal achievement while using a tractable, and thus probably imperfect, model the real world. Its plans must be efficient and goal–directed as well as sensitive to runtime variations. And it must be able to learn. One method by which these issues can be addressed is through an integration of classical planning and reactive planning. This thesis presents one such integration, in which a classical planner is augmented with the ability to defer achievable goals. Thus, a planner may now utilize any additional information which becomes available during execution to address the deferred goals.

Consider, for example, the problem of hammering a nail into a wooden plank. When we plan for this goal, we do not know how many blows it will take. We do know, however, that each blow will drive the nail further into the piece of wood and that eventually we will drive it all the way down—i.e. the desired state is achievable. Thus while we defer the computation of how many hammering actions to execute, we are nevertheless sure that we will be able to perform this computation dynamically during execution by observing the progress of the nail through the wood as we hammer it in and stopping when the top of the nail head is flush with the top face of the wooden plank. By deferring the decision of how times to pound the nail, we avoid having the reason about the density of the wood, the force of the hammer blow, the angle of impact, and a whole host of other things which would be necessary for an accurate computation. At the same time, we gain the ability to utilize the information we can obtain at runtime about the progress of the nail through the wood by monitoring the execution of our actions. Furthermore, by making sure that the deferred goal is achievable, we construct a provably-correct plan much as a classical planner would. It is precisely this kind of reasoning—knowing that certain states are achievable and deciding to deal with them during execution—that the integrated approach attempts to capture.

The integrated approach, by requiring that only achievable goals be deferred, provides a method by which completable reactive plans can be constructed. This thesis also presents an explanation-based learning algorithm for learning general plans for use in the integrated approach. Explanation-based learning strategies [DeJong86, Mitchell86] have been applied to classical planners [Chien89, Fikes72, Hammond86, Minton85] and more recently to reactive planners [Blythe89, Schoppers87], but standard explanation-based learning is not directly applicable to the planning problems attacked by the integrated approach because of the nature of the plans involved. The plans in the integrated approach may contain certain parameters whose values are determined only during execution. Standard explanation-based learning, however, does not distinguish between these parameters and those whose values are determined prior to execution, a limitation which may lead to potentially useless generalizations of reactive plans. Furthermore, a general completable plan must include operators for finding an appropriate completion during execution. The learning strategy developed in this thesis introduces the idea of conjectured variables to represent plan parameters whose values are determined only during execution as well as the idea of completors, which are responsible for finding values for conjectured variables and thus completing plans

This thesis is organized in the following manner. Chapter 2 introduces the integrated approach to planning using a formalization of the planning problem within which are also presented the solutions offered by classical planning and reactive planning. Chapter 3 discusses achievability, a central concept in the integrated approach, and presents achievability proofs for three classes of problems. Chapter 4 presents contingent explanation-based learning, an approach to learning general completable reactive plans which uses conjectured variables and deals with completors. Chapter 5 shows how an implemented system uses contingent explanation-based learning in learning a general plan for spaceship acceleration. Chapter 6 discusses related work in various areas. Finally, Chapter 7 presents conclusions and future work.

Chapter 2
AN INTEGRATED APPROACH TO THE PLANNING PROBLEM

Planning may be defined as the process of determining a sequence of actions, known as a plan, which will achieve a given goal from a particular initial state. Correctness, tractability, robustness, and flexibility are some of the major concerns in planning, and different planning approaches emphasize different concerns. Classical planning provides correctness, but requires perfect a priori knowledge. Reactive planning, on the other hand, provides real–world tractability but sacrifices guarantees of success. Each provides an approach with many desirable features, but only at the cost of imposing certain constraints, thus limiting their applicability. The integrated approach to planning introduced in this chapter combines ideas from both classical planning and reactive planning, gaining the advantages of each approach while minimizing their limitations in dealing with certain classes of real–world planning problems.

This chapter begins with a formalization of the planning problem, which is then used as a framework for discussing approaches to planning. Classical planning is discussed next, followed by reactive planning. The integrated approach to planning is then presented. Finally, the relative advantages and disadvantages of the integrated approach compared with classical planning and reactive planning are discussed.

2.1 The Planning Problem

We will use a state–based representation of plans, which characterizes the world as being in one of a discrete number of quiescent states at any one point in time, with actions being the only means by which different states can be reached. In this formalization, only a single agent and non–simultaneous actions are allowed. Although coordinating multiple intelligent agents [Cohen89, Durfee88, Tennenholtz89, Zlotkin89] and dealing with counteragents [Birnbaum86, Collins87, Konolige89] are interesting topics, they are ones outside the scope of this research. Representing overlapping actions is also an interesting problem [Allen83, McDermott82], and one not readily handled by a state–based representation. However, this limitation does not greatly affect the representational issues being addressed here, and so only non–simultaneous actions are considered.

A *state* is a snapshot of the world at a given point in time. States can be represented by state descriptions. Each *state description* S consists of a conjunctive set of sentences and corresponds to the set of states, denoted by states(S), in which S is true. In planning, an initial state description I is usually a complete description, and thus states(I) contains a single, unique initial state. However, a goal state description G is usually only a partial state description, and thus states(G) corresponds to a set of states (Figure 2–1).

An *action* is a change in the world, defined by its preconditions and effects. An action can be represented using an *action description*, which consists of an *action designator* a, a *precondition state description* PR(a),

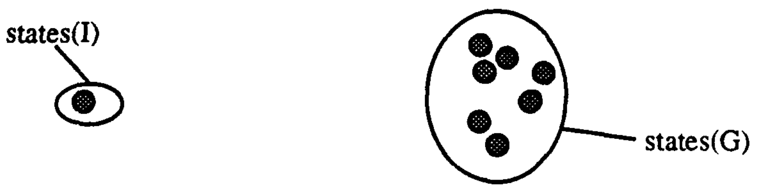


Figure 2–1. States denoted by a complete initial state description I and partial state description G.

and an *effect state description* EF(a).[1] The action designator a denotes the physical action itself, and PR(a) and EF(a), the possible precondition and effect states. If PR(a) and EF(a) are complete state descriptions, then the action description specifies a single, unique precondition state and a single, unique effect state—i.e. states(PR(a)) and states(EF(a)) consist of one element each. However, if PR(a) and EF(a) are partial state descriptions, then the action description specifies a mapping from a set of states to a set of states. This represents the knowledge that the action, when executed in some state satisfying PR(a), will result in some state satisfying EF(a) (Figure 2–2). A convenient notation for a planner's knowledge of actions is the function DO,

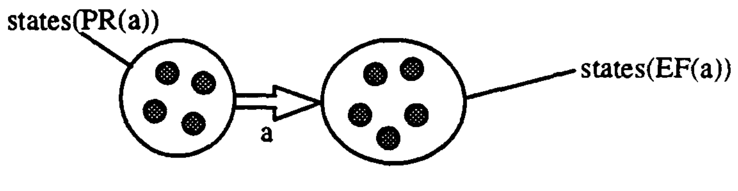


Figure 2–2. Action and states denoted by an action description consisting of an action designator a, precondition state description PR(a), and effect state description EF(a).

which maps actions and states into states. Specifically, DO takes an action and a state and maps these onto the state which results from the execution of the action in the first state. Thus:

$$\mathrm{DO}(a,s) \in \mathrm{states}(\mathrm{EF}(a)) \text{ iff } s \in \mathrm{states}(\mathrm{PR}(a))\,.$$

A *plan* is a sequence of actions. Let s be a state and p be a plan consisting of a sequence of actions $\{a_1,a_2,\ldots,a_{n-1},a_n\}$. Then DO is defined for plans in the following manner:

$$\mathrm{DO}(p,s) = \mathrm{DO}(a_n,\mathrm{DO}(a_{n-1},\mathrm{DO}(\ldots,\mathrm{DO}(a_2,\mathrm{DO}(a_1,s))\ldots))).$$

A plan p is a plan for an initial state description I and a goal state description G if executing p from any state in states(I) results in a state in states(G):

$$\text{isa-plan-for}(p,S_1,S_2) \text{ iff } [s \in \mathrm{states}(S_1) \rightarrow \mathrm{DO}(p,s) \in \mathrm{states}(S_2)].$$

1. An operator description also consists of a precondition state description, effect state description, and action designator. However, the action designator of an operator description may contain variables whereas that of an action description may not. An action is thus simply a fully–instantiated operator. The relationship between actions and operators as they relate to this thesis will be discussed further in a succeeding chapter.

Furthermore, if p is a plan from S_1 to S_2, we say p is applicable to any state in states(S_1):

isa–plan–for(p,S_1,S_2) $\rightarrow$ [s $\in$ states(S_1) $\rightarrow$ applicable(p,s)].

Alternatively, a plan p is applicable in a state s if it can be executed from that state—i.e. the preconditions of the plan are satisfied in that state.

The planning problem can now formally be stated as:

Given a complete initial state description I (i.e. states(I) = {i}) and
a partial goal state description G

Find a plan p such that DO(p,i) $\in$ states(G).

Planning may be thought of as having two phases: construction and execution. Different approaches to planning place varying emphases on these two phases. The different planning paradigms may thus be characterized by the kind of work they entail in the construction and execution of plans.

2.2 Classical Planning

Classical planning [Chapman87, Fikes71, Sacerdoti77, Stefik81, Sussman73] concentrates on the problem of plan construction. In creating a plan, a classical planner essentially creates a logical proof as to why a certain sequence of actions will achieve a specified goal. With the planner having soundly and consistently inferred the effects of a particular sequence of actions, the problem of plan execution becomes trivial. Provided the planner has perfect a priori knowledge about the world, the plan it constructs is indeed guaranteed to follow the predicted execution pattern.

Classical planning has usually been studied under the assumption of complete information, which allows the a priori determination of a complete plan for achieving a given goal. Here, we characterize one possible approach by a classical planner in solving planning problems wherein it does not have complete information, and thus must construct several alternative plans for achieving the goal from the various possible intermediate states it may reach during execution. This assumes that the classical planner will have some runtime ability for distinguishing between states to determine an appropriate subplan to execute. However, plan construction remains completely an a priori affair, with the set of plans having to be completely determined prior to execution.

Let X be a set of plans. X is said to contain a plan for the states corresponding to an initial state description A and goal state description B if for every state in states(A), there is some combination of the plans in X which when executed from the state in states(A) results in a state in states(B):

has–plan–for(X,A,B) iff

[($\exists$ x [x $\in$ X AND isa–plan–for(x,A,B)]) OR
((has–plan–for(X,A,C) AND has–plan–for(X,C,B)) OR
((has–plan–for(X,A_1,B) AND has–plan–for(X,A_2,B) AND ...
AND has–plan–for(X,A_n,B)),
where A1 $\cup$ A2 $\cup$ ... $\cup$ An = A)]

(Figure 2–3). Note that A and B may be descriptions for intermediate states within a larger planning problem. That is, planning to reach a goal satisfying G from an initial state satisfying I may involve planning to reach some intermediate state satisfying B from some intermediate state satisfying A. So as not to confuse the initial

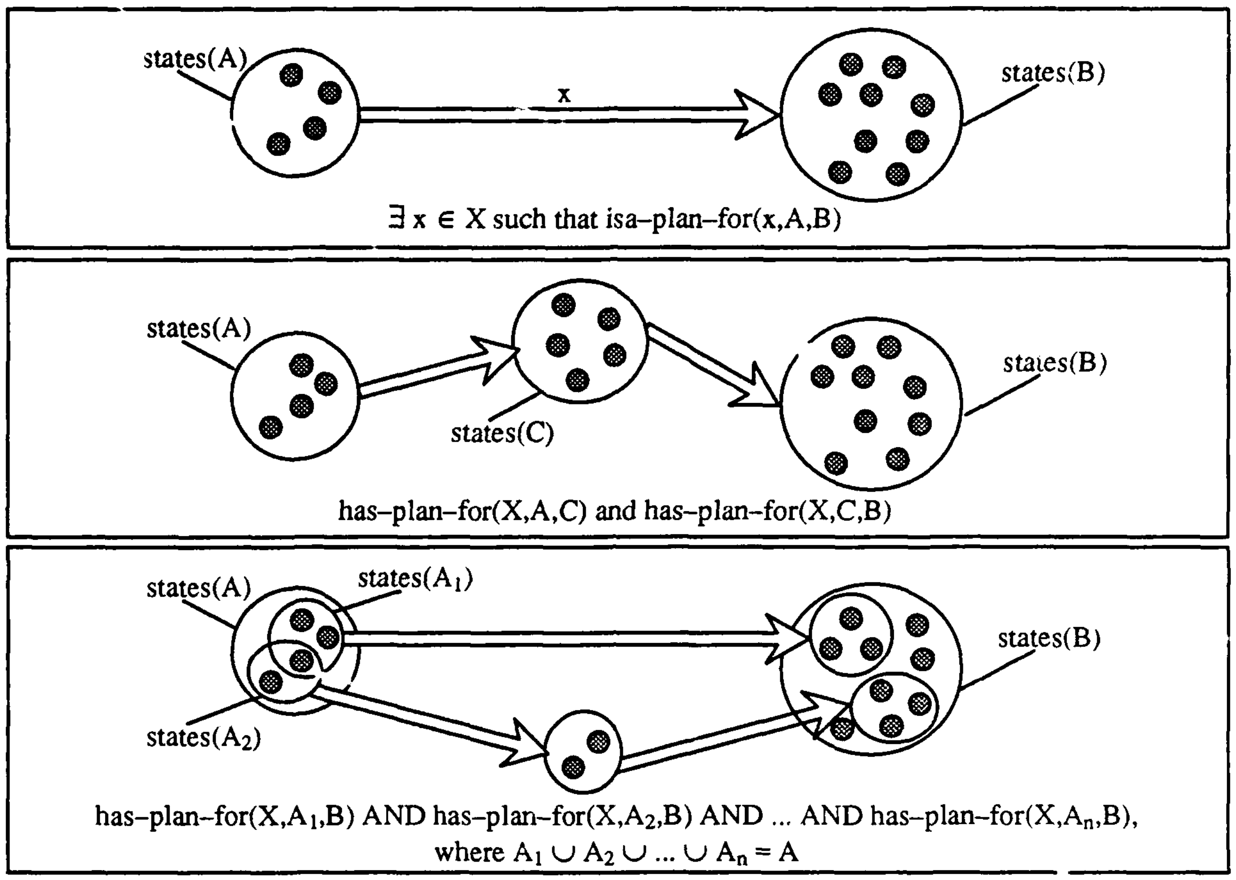

states(A)
x
states(B)
$\exists$ x $\in$ X such that isa-plan-for(x,A,B)
states(A)
states(C)
states(B)
has-plan-for(X,A,C) and has-plan-for(X,C,B)
states(A)
states(A_1)
states(B)
states(A_2)
has-plan-for(X,A_1,B) AND has-plan-for(X,A_2,B) AND ... AND has-plan-for(X,A_n,B),
where $A_1 \cup A_2 \cup ... \cup A_n = A$

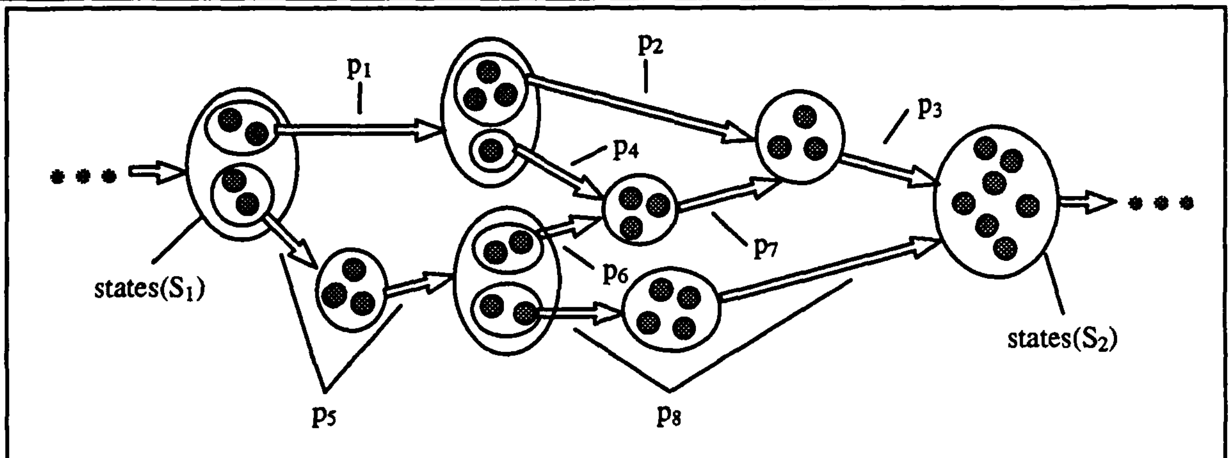
Figure 2-3. For state descriptions A and B and a set of plans X: has-plan-for(X,A,B).

state and goal state of a planning problem with a pair of intermediate states within that problem for which a subplan must be found, the terms initial state and goal state will be reserved for planning problems, and intermediate initial states and goal states will be referred to simply as *first states* and *second states* respectively.

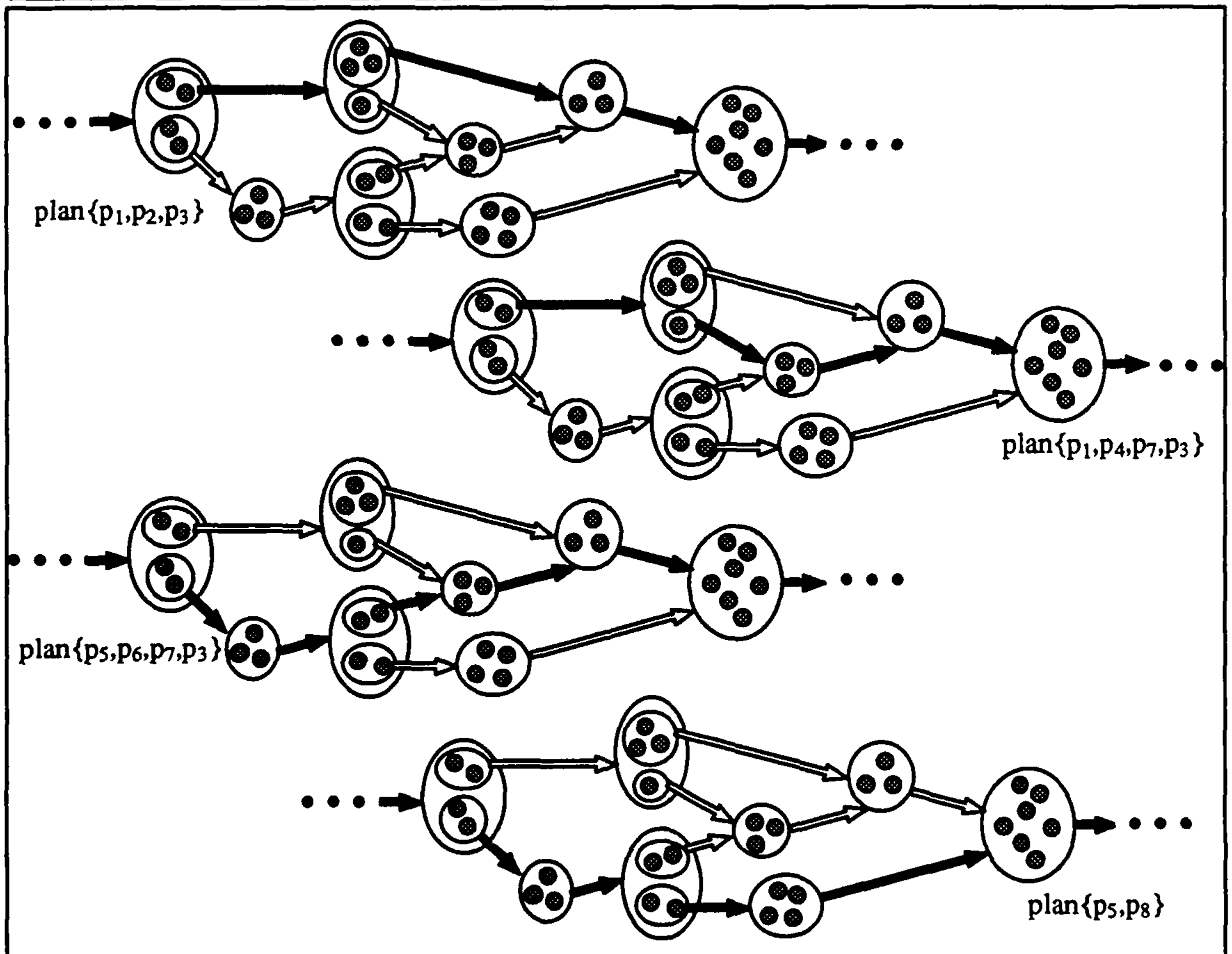
Given an initial state description I and goal state description G, the classical planning solution to the planning problem is thus:

```
Classical Planning
    Prior to Execution:
        Determine a set of plans P such that has-plan-for(P,I,G).
    At Execution Time:
        While the current state s $\notin$ states(G)
            Choose p $\in$ P such that applicable(p,s)
            Execute p.
```

Consider the following situation, in which a classical planner must plan to reach a state satisfying a second state description S_2 from the states satisfying a first state description S_1 (Figure 2-4). A classical planner might construct a set of plans X (Figure 2-4a), from which could be derived various plans to cover the different possible intermediate states (Figure 2-4b). For any particular planning problem, a classical planner may



a. $X = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.



b. Complete plans from X.

Figure 2–4. Classical planning solution.

have to determine many such plan sets X—one for each pair of first and second state descriptions S_1 and S_2 for which no single plan can be determined to achieve a state in states(S_2) from the states in states(S_1). Classical planning does not allow for planning during execution, and without the knowledge enabling the precise states which will be reached during execution to be determined, classical planning has no recourse but to exhaustively consider all the possibilities in a priori determining a set of plans covering all the intermediate states which may be reached during execution.

Unfortunately, while such a laborious task may be possible in some well–behaved microworld, it is infeasible if not impossible in any real world domain. The power of classical planning lies in its ability to a priori determine provably–correct plans—that is, plans which are guaranteed to achieve the goal. However, with inference as its sole guide in plan construction, the classical planner is faced with the extended prediction problem [McCarthy69, Shoham86], which is the problem of determining what aspects of the world remain unchanged by the execution of an action. Even with perfect domain theories, classical planning has been proven undecidable and intractable [Chapman87]. Thus, while classical planning guarantees successful execution, it does so only at the cost of real–world applicability.

2.3 Reactive Planning

Reactive planning [Agre87, Firby87, Schoppers87, Suchman87] provides one answer to the problems faced by classical planners in real–world domains. Reactive planning, in the extreme, proposes that the actions of intelligent agents are actually mere reactions to stimuli provided by the real world. By not concerning itself with the future, by always determining its next action based on the current state of the world, the reactive planner avoids having to determine what will be true of the world after the execution of an action. Thus, reactive planning solves the extended prediction problem simply by eliminating it.

Earlier, the idea of applicability for plans and states was introduced as follows:

$\text{isa–plan–for}(p,S_1,S_2) \rightarrow [s \in \text{states}(S_1) \rightarrow \text{applicable}(p,s)]$.

That is, a plan p for an initial state description S_1 and goal state description S_2 (both possible referring to intermediate states) is applicable to any state satisfying S_1. Since an action is also a plan from its precondition state description to its goal state description—i.e. isa–plan–for(a,PR(a),EF(a))—applicability can also be defined for actions and states as follows:

$\text{applicable}(a,s) \text{ iff } s \in \text{states}(PR(a))$

or equivalently, since [$DO(a,s) \in \text{states}(EF(a)) \text{ iff } s \in \text{states}(PR(a))$]:

$\text{applicable}(a,s) \text{ iff } DO(a,s) \in \text{states}(EF(a))$.

That is, an action is applicable in a state if its preconditions are satisfied in that state.

Given an initial state description I and goal state description G, the reactive planning solution consists solely of an execution–time phase wherein a plan is dynamically determined:

```
Prior to Execution
    <nothing>
At Execution Time
    While the current state $s \notin$ states(G)
        Choose a such that applicable(a,s).
```

Although similar, the execution phase in reactive planning is much less restricted than that of classical planning. While a classical planner is bound to choosing a plan from a predetermined set of plans, a reactive planner has the flexibility to choose any executable action. A reactive planner thus not only saves itself the effort of constructing plans which will never be used, but it may also able to deal with unforeseeable circumstances which would cause a classical planner to fail.

Unfortunately, because no predictions about the future are made, successful plan execution cannot be guaranteed either. The reactive planner lives one action at a time and makes irreversible planning decisions without considering alternatives beyond their immediate applicability to the current state. Thus, plan execution may often stray from the path of the ideal plan, resulting in a sub–optimal path towards the goal, or even the inability to reach the goal (Figure 2–5). Because reactive planning is essentially a hill–climbing approach,

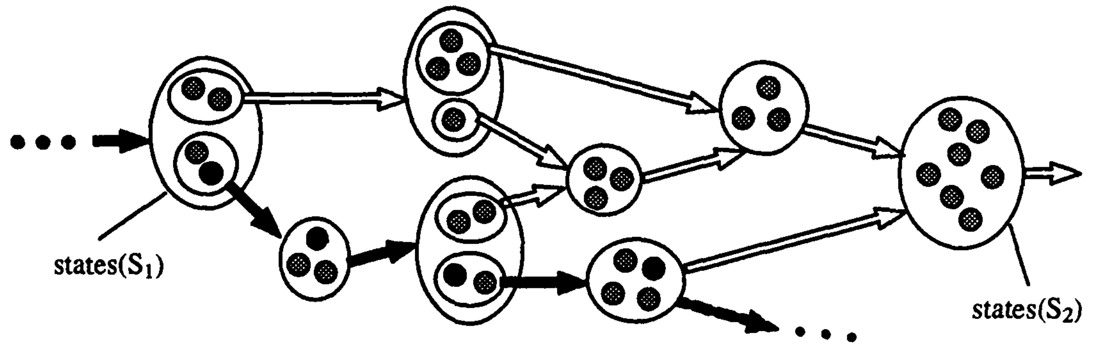


Figure 2–5. A possible wrong choice made by a reactive planner leading to a sub–optimal or failed plan.

one wrong decision could preclude the goal ever being reached.

Reactive planners also lack explicit concepts about goals and goal achievement beyond a single evaluate–act cycle, and consequently cannot reason about interacting effects among groups of actions and their consequences upon goal achievement. Thus, the rules a reactive planner uses to determine appropriate actions must be hand–tailored for every domain to achieve the desired behavior. Consequently, current reactive planners cannot easily extend their problem–solving capabilities. Whereas machine learning strategies have been successfully applied to classical planners in various domains [Chien89, Fikes72, Hammond86, Minton85], only preliminary work has been done on learning reactive rules [Blythe89, Schoppers87].

Reactivity is good because it allows a system to respond to various conditions of the world which would be very difficult to predict a priori. However, if reactivity is the sole means by which a system can act, then goal–directed behavior can only be possible if it is built into the rules the system uses to decide on its actions. Purely reactive planners may be the right answer in very unpredictable domains for which a correct and tractable domain theory is difficult to construct. They can also be more effective in dynamic environments in which planning tasks arise at different times and interact in complex ways. However, many domains can be characterized by tractable domain theories reasonably well, and many planning situations allow a priori reasoning. In these cases, the benefits made available by reactivity may not be worth the accompanying drawbacks.

2.4 An Integrated Approach

An obvious approach to solving the limitations of both classical planning and reactive planning while gaining their benefits is to integrate the two. The classical planner is attractive because it constructs provably-correct plans, uses domain-independent planning strategies, and can readily incorporate machine learning strategies in its problem-solving. However, classical planning breaks down in complex domains where the requisite perfect domain theory is either unavailable or intractable. The reactive planner which dynamically decides upon its actions during execution is able to handle complex, rapidly-changing domains. However, it lacks the goal-directedness, guarantees of success, and domain independence afforded by classical planning.

In the integrated approach, the classical planner is augmented with the ability to defer goals. To maintain the provably correct nature of the plans constructed, the following restriction is imposed: a goal may be deferred only if it can be proven achievable, where *achievability* is defined between states as follows:

achievable(S_1,S_2) iff [$s \in$ states(S_1) $\rightarrow \exists$ p such that DO(p,s) $\in$ states(S_2)].

Note that achievability involves only proving the *existence* of a plan p and not necessarily determining p itself. While determining a plan p to achieve the goal is certainly one way of proving achievability, it is not the only way to prove achievability. There are certain classes of problems for which achievability proofs can be constructed independently of determining plans to achieve the associated goals, and these are precisely the kinds of problems addressed by the integrated approach. The concept of achievability is a central notion in the integrated approach, and the next chapter is devoted to its discussion. Here, the overall solution to the planning problem offered by the integrated approach is first presented.

The classical planning approach involved the determination of a set of plans P for an initial state description I and a goal state description G such that has-plan-for(P,I,G). The integrated approach also involves the a priori determination of a set of plans P, but imposes a different constraint on this set of plans which allows achievability proofs in addition to plans. A set of plans X is considered to be a set of plans for an initial state description A and goal state description B if there exists some combination of subplans in X and/or achievability proofs achieving a state in states(B) from every state in states(A):

has-plan-for-or-achievable(X,A,B) iff

[($\exists$ x [x $\in$ X AND isa-plan-for(x,A,B)]) OR
(has-plan-for-or-achievable(X,A,C) AND has-plan-for-or-achievable(X,C,B)) OR
(has-plan-for-or-achievable(X,A_1,B) AND has-plan-for-or-achievable(X,A_2,B)
AND ... AND has-plan-for-or-achievable(A_n,B,X),
where $A_1 \cup A_2 \cup ... \cup A_n$ = A) OR
(achievable(A,B))].

Thus, rather than determine a set of plans P such that has-plan-for(P,I,G), a planner in the integrated approach determines a set of plans P such that has-plan-for-or-achievable(X,A,B). The only difference between classical planning and the integrated approach is the option of proving achievability instead of determining a sequence of actions between states in the integrated approach. This difference, however, may lead to a substantial difference in the size of the set of plans constructed by a classical planner and a planner in the integrated approach. For example, consider again the situation in Figure 2-4. Instead of the set of eight subplans constructed by a classical planner, the solution of a planner in the integrated approach may instead be a combination of achievability proofs and a small subset of these plans (Figure 2–6).

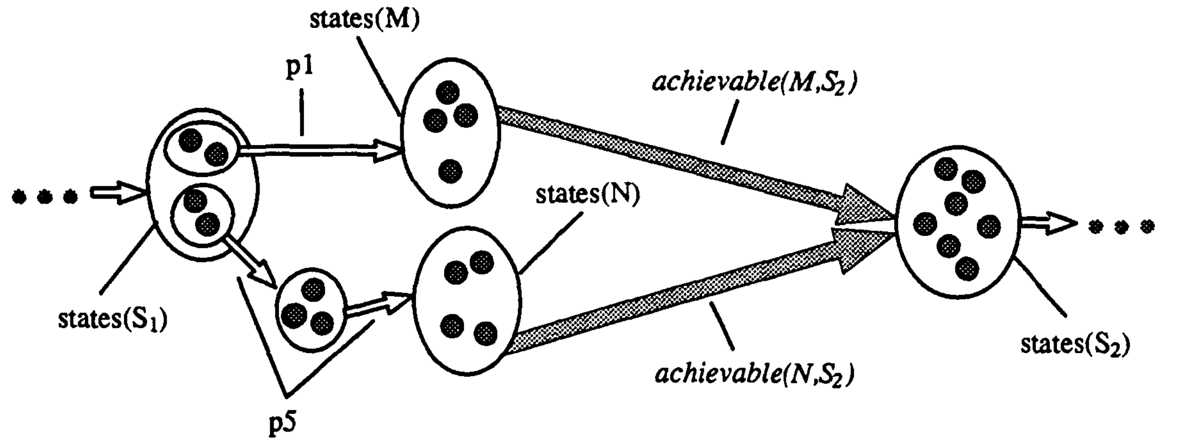


Figure 2–6. Integrated approach: has–plan–for–or–achievable(X,S_1,S_2); $X = \{p_1,p_5\}$.

Since the set of plans constructed prior to execution yields only incomplete or partial plans, during execution additional planning decisions must be made by a planner in the integrated approach to complete a plan to achieve the goal. Performing this completion step during execution has the advantage that additional information may become available for discriminating between the different possible states and focusing attention towards a particular plan. For example, by the time the planner reaches a state in states(S_1) during execution, additional information may enable it to choose plan p_5 over plan p_1. Furthermore, upon reaching a state in states(N), it will have the benefit of runtime information in constructing a plan from the state reached in states(N) to a state in states(S_2)—a plan guaranteed to exist by the a priori proof that achievable(N,S_2) (Figure 2–7). Like reactive planning, the integrated approach saves a planner from constructing some plans

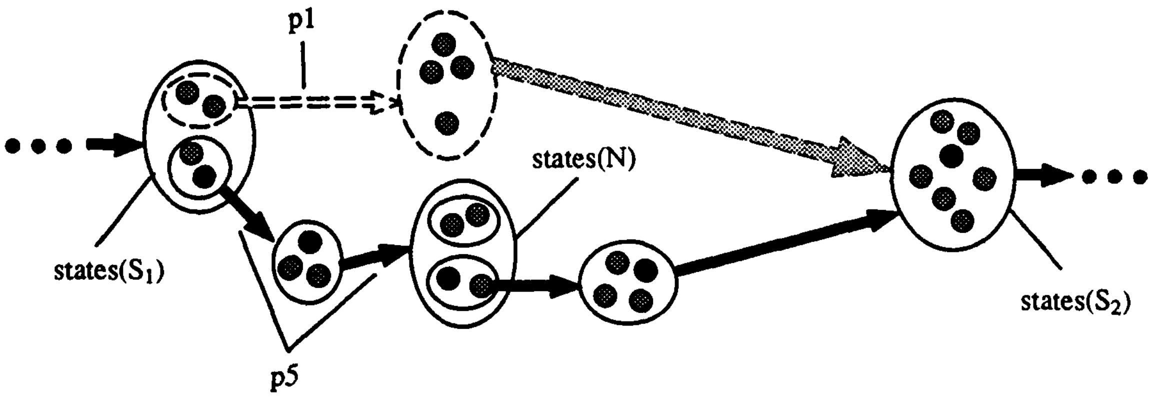


Figure 2–7. Completed plan from states(S_1) to states(S_2). Dark states represent the states reached during execution; bold arrows represent the plan.

which will never be used. Unlike in reactive planning, however, these savings do not come at the cost of the provably–correct nature of the plans being constructed.

In summary, given an initial state description I and goal state description G, the integrated approach to planning consists of the following two steps:

```
Prior to Execution
    Determine P such that has–plan–for–or–achievable(P,I,G).
At Execution Time
    While the current state s $\notin$ states(G)
        If  $\exists$ p $\in$ P such that applicable(p,s)
            then
                Execute p
            else {no applicable plan was determined a priori, hence s $\in$ states($S_1$),
                 where achievable($S_1$,$S_2$)}
                Determine p such that DO(p,s) $\in$ states($S_2$).
```

2.5 Analysis

The integrated approach augments a classical planner with the option to defer achievable goals, and the primary differences between the two approaches stem from this additional feature. A classical planner constructs a complete set of plans prior to execution, whereas a planner in the integrated approach has the option of constructing partial plans to be completed during execution but with the additional cost of constructing achievability proofs. The execution–time phase of the integrated approach will also generally be more complicated than that of classical planning since it may involve plan construction. However, since additional execution–time information may enable a planner in the integrated approach to narrow down the number of states under consideration, the cost of dynamic plan construction will also generally be less than the cost incurred by a classical planner in constructing a corresponding set of plans, which will have to cover all possible execution sequences. Thus, since both classical planning and the integrated approach provide proofs of correctness for the plans they construct, the integrated approach will win over classical planning in those cases wherein the combined cost of the achievability proofs and dynamic plan construction is less than the cost of determining the corresponding set of plans in classical planning.

Reactive planning performs no planning prior to execution and hence the integrated approach has that additional cost over reactive planning. However, machine learning strategies such as those which have been applied to classical planning [Chien89, Fikes72, Hammond86, Minton85] may enable the cost of constructing a combination of achievability proofs and a set of plans for any particular problem to be amortized over the number of times the generalization of that problem is encountered. The execution phase of the integrated approach may also be more expensive if the constraints imposed by a priori planning decisions complicate execution–time decision–making, although the execution phase of reactive planning may be the more costly one if a priori planning decisions in the integrated approach help confine the search for an executable action. The constraints placed by a priori planning on execution–time planning in the integrated approach thus has its advantages as well as disadvantages. While the integrated approach provides a planner with some ability to utilize execution–time information and adapt its plans accordingly, it will still be unable to handle certain kinds of unexpected events. However, a distinct advantage that the integrated approach has over reactive planning is the construction of prc vably–correct plans guaranteed to achieve their goals.

The integrated approach is not the perfect solution to all planning problems, however it is a solution to an interesting subset of these. In particular, for those problems in which achievability proofs can be constructed independently of determining corresponding plans, at a greatly reduced cost in plan construction, the integrated approach provides a tractable solution which provides runtime sensitivity without sacrificing correctness. In the following chapter, such problem classes and their associated achievability proofs are discussed in further detail.

Chapter 3
PROVING ACHIEVABILITY

How can a proof of achievability be constructed? That is, how can it be proven that there exists some action or sequence of actions which can be determined during execution and executed to achieve a particular deferred goal? Finding a plan to achieve the goal is one way of proving achievability, but this would entail the same exhaustive search through the space of all possible execution sequences that burdens the classical planner. Thus, proving achievability must be done independently from determining the actions to achieve the deferred goals. The determination of these actions is deferred until execution when more discriminating information is available.

This chapter begins with a discussion of some knowledge representation issues. The interval-based representation used in representing the system's knowledge, and the simple forms of qualitative reasoning the system is capable of is first introduced. Then an informal presentation is made of certain classes of problems for which achievability proofs can be constructed independently of achieving plans. Finally, three different kinds of achievability proofs are formally presented in the form of rule schemata.

3.1 Knowledge Representation

One of the initial domains studied and the one eventually chosen for implementation was a domain involving the process of acceleration. To better represent the time-spanning nature of processes, an interval-based representation scheme was chosen. Simple forms of qualitative reasoning were also incorporated to provide a higher grain-size of representation for reasoning about processes, which enabled the construction of the kinds of achievability proofs sought in the integrated approach to planning.

3.1.1 Interval-Based Calculus

The system's knowledge is represented in a simplified interval-based calculus. Predications can be made over time points as well as over intervals, with a predication over an interval necessarily being made over the largest possible interval. An interval is represented by a pair of time points designating its starting and ending points. The starting point defaults to the start of time, T–INF, and the ending point to the end of time, T+INF. For simplicity, predications can also be made without reference to an interval, in which case they are essentially always true (i.e. true over the interval (T–INF T+INF)).

Operators have preconditions and direct effects as well as inferred effects. The direct effects are incorporated into the operator definition. The inferred effects are implemented as intra-situational rules. Both the operators and rules may introduce interval constraints, which are maintained implicitly during the explanation process. Actions always occur over intervals.

3.1.2 Qualitative Reasoning

Processes are time-spanning events which, if modeled completely, would result in a very complex, intricate domain theory likely to be computationally intractable. Qualitative reasoning [deKleer84, Forbus84, Kuipers84, Williams84] provides a larger grain-size or qualitative level at which to reason about physical systems. For example, instead of understanding the process of heating water in tenns of energy transfer and excitation of atoms, a qualitative-reasoning system might understand it as the application of heat to water which causes the temperature of the water to rise.

The system's qualitative reasoning ability is couched in an adaptation of Qualitative Process Theory [Forbus84]. The system has knowledge of quantities. A quantity can have an instantaneous value—i.e. a predication of its value made over a time point—as well as a monotonic behavior over time intervals—i.e. increasing, decreasing, or constant. Each quantity is also associated with its derivative, or rate of change. The quantity of acceleration, for example, is the derivative of the quantity of velocity. A quantity may also be qualitatively proportional, either positively or negatively, to other quantities. For example, acceleration is positively qualitatively proportional to net force—i.e. the larger the net force, the higher the acceleration rate.

In Qualitative Process Theory, quantities may be characterized as being independent entities or existing only in the context of a process. Here we take the view of quantities being independent entities, and furthermore, being related to objects. Thus, quantities are represented as functions with the following form:

(<quantity name> <object>).

For example, the quantities representing the speed of the starship Enterprise, its acceleration rate, and the net force upon the starship may be represented by the following:

(velocity Enterprise)

(acceleration Enterprise)

(net_force Enterprise).

A quantity may take on different values at different times. Statements about a quantity's value at a particular time are of the form:

(value <quantity> <value> <time point>)

Thus, the fact that the Enterprise was traveling at 200 velocity units at time 35 in the current time frame may be represented by:

(value (velocity Enterprise) 200 35).

A quantity may also take on different time-spanning behaviors. Predications on a quantity's monotonic behavior over time are of the following form:

(qualitative_behavior <quantity> [increasing/decreasing/constant] <time interval>)

For example, if the Enterprise was accelerating over the time interval beginning at 35 and ending at 70, then the following predication is true:

(qualitative_behavior (velocity Enterprise) increasing (35 70)).

Derivative, positive qualitative proportionality, and negative qualitative proportionality relationships are represented as follows:

(derivative <quantity> <derivative of quantity>)

(q+prop <quantity-1> <quantity-2>)

(q-prop <quantity-1> <quantity-2>).

For example, the rate of change of the velocity is acceleration, and acceleration is positively qualitatively proportional to net force, hence:

(derivative (velocity Enterprise) (acceleration Enterprise))

(q+prop (acceleration Enterprise) (net_force Enterprise)).

3.2 Planning Problems with Deferrable Achievable Goals

There are three classes of problems which have been identified wherein achievability proofs can be constructed independently of determining the actions to achieve the associated deferred goals. The first class consists of problems involving repeated actions and terminal goal values; the second, continuous monotonically–changing processes and intermediate goal values; and the third, multiple opportunities. In all three, achievability proofs may be constructed for certain goals, allowing these goals to be deferred until execution time without sacrificing success while allowing certain planning decisions to be deferred until more information becomes available.

3.2.1 Repeated Actions and Terminal Goal Values

One class of problems for which a proof of achievability can be constructed independently of determining the actions to achieve the goal is the class which involves repeating a particular action an indefinite number of times to achieve a terminal goal value. Examples are hammering a nail all the way into a piece of wood, completely unloading a clothes dryer, and painting a wall. All these problems are characterized by a single action or action sequence—pound, unload, brush—performed repeatedly until a state is reached where the execution of the particular action or action sequence has no further effect on a particular goal value. Each instance of an action furthers the progress towards the goal until the goal is reached. For example, every sweep of the paint brush covers part of the wall with paint until all the whole wall is painted.

The proof of achievability for this class of problems lies in this notion of incremental progress towards a terminal goal value. Such a proof does not require finding the precise actions for achieving the goal—such as the number of unloading actions to empty the dryer or which clothes to take out in what order. It relies only on the idea of every action moving closer to the goal by at least a certain amount and thus the goal eventually being reached. If at plan construction time it can be proven that there exists some value n such that executing a particular action of action sequence n times will achieve a goal state, then determining precisely what n is may be deferred until execution time. At plan execution time, this can be done simply by performing the action, evaluating the results, and repeating this process until the goal conditions are met. Since the existence of n was proven at plan construction time, it is guaranteed that during execution, the action can be performed some finite number of times to achieve the goal.

3.2.2 Continuous Processes and Intermediate Goal Values

A second class of problems for a which an achievability proof can be constructed involves continuous processes and intermediate goal values. Some examples are beating egg–whites, freezing ice–cream, and accelerating to a higher velocity. Each of these includes a process—beating, cooling, moving faster—which involves the continuous change in the value of some quantity—the increase of air incorporated into the egg–whites, decreasing temperature of the ice–cream mixture, increasing velocity—until a particular desired qualitative state is reached—ridges begin to form, slush, and a particular speed.

As with the previous class of problems involving repeated actions, the proof of achievability for this class of problems lies in the notion of incremental progress. However, additional knowledge about continuous change enables reasoning about the achievability of intermediate values. If at plan construction time some value can be projected as the result of a particular process such that the goal quantity value lies between that value and the initial value, then the achievability of an intermediate goal value can be proven. For example, since stiff peaks will form if the egg whites are beaten long enough, beating until soft peaks form can be achieved. Such an achievability proof does not rely on determining precisely how long a particular process must run, but it ensures that there exists some interval over which if the process runs, the goal value will be achieved.

3.2.3 Multiple Opportunities

The third class of problems for which there exists an achievability proof independent of determining the actions to achieve the goal is the class which involves the existence of multiple objects in the world. The basic idea is that a particular action may be applied to any of these objects with the same desired effects. For example, the cup in getting a cup of coffee may be any of a number of cups in the cabinet. The gas station one chooses at which to fill up the tank on a trip from Champaign up to Chicago may be any of the dozens along the way. The latest adventure thriller one buys at a bookstore may be any one of the copies stacked on a table. One cup or station or book is as good as the next in achieving one's goal of drinking coffee or getting gas or relaxing one weekend.

The proof of achievability for this class of problems relies on proving the existence of several opportunities to instantiate an action, all of which are equally good at achieving the desired goal. In constructing a plan, it is sufficient to prove that the set to be chosen from is non–empty. Thus, proving achievability does not necessitate finding the exact individuals with which to instantiate a particular action sequence in order to achieve the goal. It relies simply on knowledge about certain regularities in the world. If at plan construction time it can be proven that there exist several objects, any one of which can be used to achieve the goal, then determining precisely object to use can be deferred for execution. At plan execution time, this can be done by evaluating the world and choosing an appropriate object with which to achieve the goal.

3.3 Achievability Proofs

In this section, the three types of achievability proofs corresponding to the three classes of problems in the previous section are discussed. Each type of achievability proof is presented in the form of a second–order predicate calculus rule, which serves as a schema or template by which to derive first–order predicate calculus rules to be used by the system. Current work is investigating other planning problems in an attempt to further identify other classes of problems for which achievability proofs can be constructed independently of determining the actions to achieve the associated goals. It is expected that the planning problems we are interested in can be categorized into a relatively small number of classes based on their corresponding achievability proofs.

3.3.1 Repeated Actions and Terminal Goal Values

One kind of achievability proof which has been investigated involves reasoning about repeated actions and terminal values. The achievability of terminal values for a particular quantity can be proven if it is known that the execution of a particular action results in non–asymptotic progress towards the terminal value until that terminal value is reached, from which time executing the action does not change the value of the quantity. For example, consider the problem of hammering a nail all the way into a wooden plank—i.e. driving the nail in until the top of its head is flush with the top face of the plank. In this example, the quantity being measured is the length of the part of the nail embedded in the wood, with the goal value being the length of the whole nail. Every pound of the hammer will drive the nail further into the wood until the nail is already all the way into the wooden plank. Additional blows then will not further embed the nail into the wood. Therefore, there exists some number of hammer blows which will drive the nail completely into the piece of wood. This reasoning is depicted graphically in Figure 3–1.

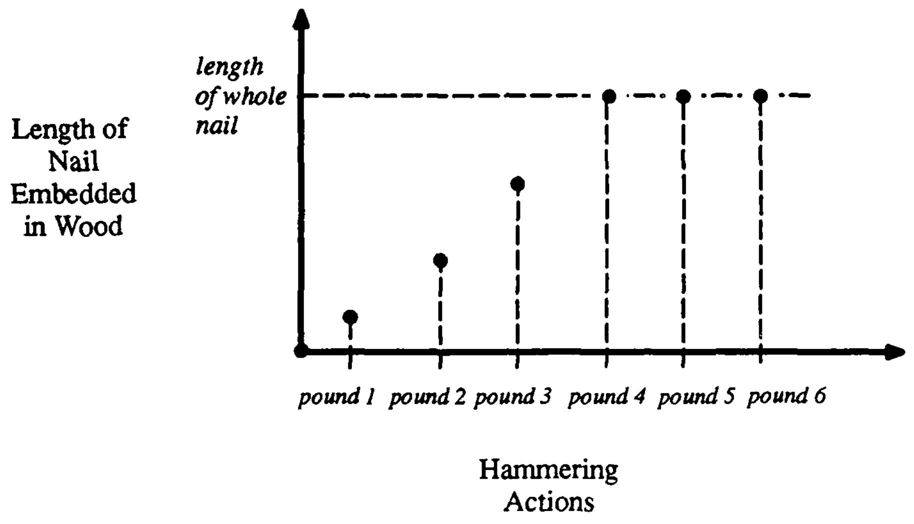


Figure 3–1. Reasoning about repeated actions and terminal goal values.

This reasoning can be represented as follows. Let q be some quantity such that the goal is to let q have the value vg. Also let (α n int) be true if the execution of a particular action or action sequence α n times in succession takes place over the interval int. The goal value vg is achievable through the repetition of α if the following three conditions hold:

1. if $q \neq vg$ then the execution of α will result in a new value for q closer to vg
2. if $q = vg$ then the execution of α will not affect the value of q
3. a change in the value of q from an execution of α does not imply that a later execution of α will result in a smaller change in the value of q (provided the distance to the goal just prior to the later execution is not less than the change from the earlier execution, since this would indeed limit the change from the later execution to an amount smaller than the change from the earlier execution).

The third condition simply guarantees that a sequence of α's will not have a decreasing effect on the value of q.[1] Together, these three conditions ensure adequate progress towards the goal, supporting a proof of achievability. A rule schema for this proof of achievability is the following:

```
Terminal Value Rule Schema
[∀ α
    [∀ q v1 v2 v3 v4 vg t1 t2 t3 t4
        (   (((value q v1 t1) AND (α 1 (t1 t2) AND (value q v2 t2)) AND (≠ v1 vg))
            → (between v2 v1 vg))
            AND
            (((value q v1 t1) AND  (α 1 (t1 t2) AND (value q v2 t2) AND (= v1 vg))
            → (= v2 vg))
            AND
            (NOT  (((value q v1 t1) AND  (α 1 (t1 t2) AND (value q v2 t2) AND
                   (value q va ta) AND (α 1 (ta tb)) AND (value q vb tb) AND
                   (after (t1 t2) (ta tb)) AND (< (– vb va) (– tg t1)))
                  → (< (– v2 v1) (– vb va)) ) ) )
    →
        [∃ n tn  ((α n (t1 tn) → (value q vg tn))] ] ].
```

In the hammering example, the first condition is true since if the nail is not yet completely in, each hammer blow will drive the nail further in.[2] The second condition is also true since if the nail is already completely in, further hammer blows will not drive it in any further. Finally, the third condition is true since the amount a nail is driven in by a particular blow is not an upper limit on the change which results from any succeeding blow. Thus, there exists some number of hammer blows which will result in the nail being driven completely into the wood.

3.2.2 Continuous Processes and Intermediate Goal Values

Another kind of achievability proof involves reasoning about intermediate goal values for continuous processes. The achievability of intermediate values for a continuously–changing quantity can be proven through the use of the limits on the value of the quantity. For example, if the airconditioning is capable of bringing the room temperature down to 60°F provided that the outside temperature is no more than 110°F, then if it is 95°F outside, any room temperature down to 60°F can be achieved. That is, since temperature changes continuously, for every intermediate value, there will exist some point in time at which the temperature will have that value.

1. This is an overly strong constraint, since a converging sequence of effects on the value of a quantity may still allow for a goal value to be reached. However, this constraint captures much of the intuitive reasoning about repeated actions and terminal values without sacrificing correctness, hence the use of it here. While completeness has its merits, the primary focus of this research is not in developing all–encompassing, airtight achievability proofs but in showing that there exist ways of proving achievability for certain classes of problems.

2. Barring, of course, a knot in the wood or a misdirected hammer blow. In the examples discussed here, we will assume relatively well–behaved circumstances and plan only for such situations.

Let q be a continuous quantity having a value v0 at some time t0. Also let is be the case that certain conditions θ being true over some interval (t0 t2) will result in q having some greater value v2 at time t2. Then for all values v1 between v0 and v2 there exists some time t1 within the interval (t0 t2) such that if θ holds over the interval (t0 t1), q will have the value v1 at t1.[3]This reasoning is depicted graphically in Figure 3-2. A rule

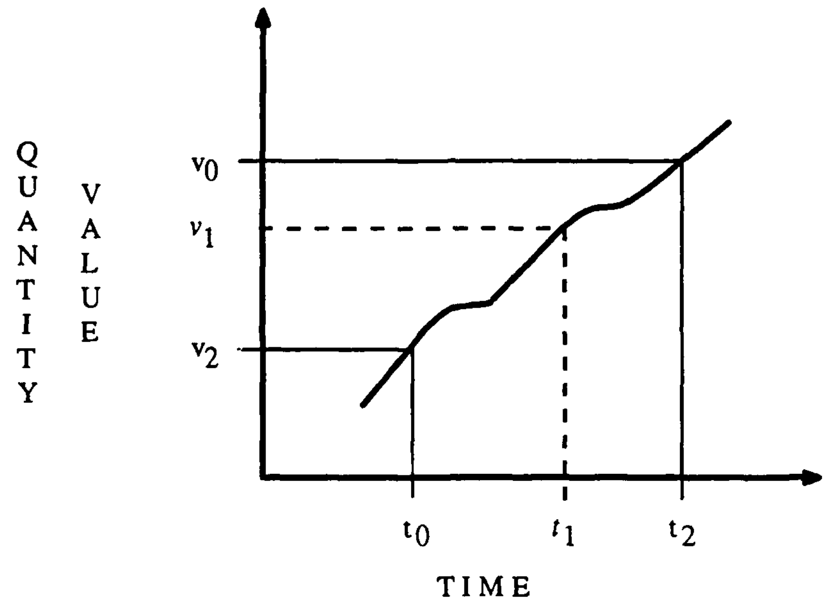


Figure 3-2. Reasoning about continuous processes and intermediate goal values.

schema for the achievability proof supported by this reasoning is the following:

```
Intermediate Value Rule Schema
[∀ θ
    [∀ q v0 v2 t0 t2
        (   (value q v0 t0) AND
            (continuous q) AND
            ((θ (t0 t2)) → (value q v2 t2))  )
    →
        [∀ v1
            ((between v1 v0 v2)
            →  [∃ t1 ((within t1 (t0 t2)) AND ((θ (t0 t1)) → (value q v1 t1)))] ) ] ] ].
```

In the cooling example, θ is that the room temperature is decreasing. This is true because the airconditioning is on and the outside temperature is less than 110 degrees. If the airconditioning remains on long enough, the lower limit of 60 degrees will be reached. Thus, if the goal is to achieve a room temperature of 75 degrees

3. This is also known as the Intermediate Value Theorem of Calculus, which states that "If N is any number between f(a) and f(b), then there is at least one number c between a and b such that f(c)=N." [Thomas68]

from the current 85, then it is guaranteed that at some time t1 the room temperature can be brought down to the goal value.

3.3.3 Multiple Opportunities

A third type of achievability proof which has been studied is one which concerns the availability of several suitable objects upon which to execute an action in order to achieve a goal. If it is known that several such objects exist, then it is unnecessary to decide at plan construction time which object will be used. Consider the problem of getting a cup of coffee. If it is known that the cupboard in the lounge is kept well-stocked with paper cups, among other things, then it can be proven that an empty, graspable cup will be available at execution time.

This reasoning can be represented as follows. Let (β int) be true if the goal β is true over the interval int, and let (α obj int) be true if a particular action or action sequence or action sequence α involving obj is executed over the time interval int. Also let τ be some set of conditions which determines whether an object is of a particular type, and let (collection objs (t1 t2)) be true if there exists a non-empty set of objects objs over the interval (t1 t2). Then if executing α on some object of satisfying the conditions τ results in the goal β being true, and there is some group of objects all satisfying τ, then there is at least one object satisfying τ with which α can be performed to achieve β. A rule schema for this proof of achievability is the following:

Multiple Opportunities Rule Schema

$$\begin{array}{l}
[\forall\ \alpha\ \beta\ \tau \\
\quad [\forall\ \text{obja objb objs t1 t2 t3} \\
\qquad (((\tau\ \text{obja})\ \text{AND}\ (\alpha\ \text{obja (t1 t2)})) \rightarrow (\beta\ \text{(t2 t3)}))\ \text{AND} \\
\qquad \text{(collection objs (t1 t2))}\ \text{AND} \\
\qquad \text{((member objb objs)} \rightarrow (\tau\ \text{objb}))\ ) \\
\quad \rightarrow \\
\qquad [\exists\ \text{objx ((member objx objs)) AND} \\
\qquad\qquad ((\tau\ \text{objx})\ \text{AND}\ (\alpha\ \text{objx (t1 t2)})) \rightarrow (\beta\ \text{(t2 t3)}))\ )\ ]\ ]\ ].
\end{array}$$

In the coffee example, an object satisfies τ if it is a paper cup which can be used to contain liquid and from which the liquid can be drunk, and objs is the stack of paper cups in the cupboard. Since there exists the collection objs all satisfying τ, there exists some object objx satisfying τ. Thus objx can satisfy both the precondition that a containing vessel exist for pouring coffee into, as well as the goal condition that a drinking vessel contain the coffee to be drunk.

The multiple opportunities, intermediate value, and terminal value rule schemata are domain-independent templates from which domain-specific first-order rules can be derived for use in proving achievability. Aside from abstracting a particular strategy for proving achievability, each schema also represents a wide class of problems for which proving achievability can be done independently of determining precisely how to achieve the associated deferred goals. This independence permits such goals to be achieved during execution without sacrificing guarantees of success, thus maintaining the notion of provably-correct plans from classical planning while gaining the flexibility of reactive planning in using execution-time information in addressing planning goals. Other problem classes for which similar rule schemata might be constructed include those involving repeated actions and intermediate goals, continuous actions and terminal goals, or choosing from several actions which can each achieve the goal. While these classes of problems are very similar to those discussed in this chapter, each varies problem conditions enough to make the previously presented achievability proofs inapplicable. A discussion on how other types of achievability proofs might be made possible is presented in the chapter on future work.

By allowing proofs of achievability to stand in for certain portions of a plan prior to execution, plans constructed a priori can be made much simpler, involving fewer actions and fewer constraints. By predetermining the planning goals which must be made during execution, goal–directedness can be maintained even as the planner has to rely on execution–time information to complete its plans. Additionally, just as different goals are addressed at different levels in hierarchical planning [Sacerdoti77, Stefik81], more efficient planning is achieved in the integrated approach through the deferment of certain goals until execution, when additional information may be gathered to aid in making planning decisions. The integrated approach to planning thus provides a simple, goal–directed, and efficient planning solution for those classes of problems wherein proving achievability can be done independently of determining the actions to achieve the associated deferred goals.

Chapter 4
LEARNING REACTIVE PLANS

The planning problem and the solutions of classical planning, reactive planning, and the integrated approach were discussed in a previous chapter. The basic idea behind the integrated approach is to reap the benefits of both classical and reactive planning, at the same time solving some of their limitations. One of the advantages of classical planning is its compatibility with various machine learning strategies. This enables the classical planner to expand its problem–solving capabilities. The integrated approach retains this capability, however certain modifications have to be made to learning techniques in order to make them applicable to reactive plans.

In this chapter, an explanation–based learning method for learning reactive plans is presented. First, a presentation of explanation–based learning as applied to learning plans is made, together with the problems reactive situations pose for standard explanation–based learning. Next, a modified explanation–based learning algorithm called contingent explanation–based learning is introduced through the ideas of conjectured variables and contingent explanations. The final section discusses different types of completors which a planner can use to complete reactive plans during execution.

4.1 Learning General Plans

Earlier, a plan was defined as a sequence of actions. Here we now extend the definition of a plan to operators. Consider the actions of stacking block a on block b, stacking block c on block d, and picking up block e. The three are different actions, but there is a certain commonality between the first two actions not shared by the third. This is exactly the type of commonality captured by operators. An *operator* conceptualizes a generic act by specifying a function from objects to actions [Genesereth87]. Thus, the STACK operator applied to block a and block b denotes the action of stacking block a on block b. Applied to other blocks, it specifies other stacking actions.

A *general plan* is defined to be a sequence of operators and a set of codesignation constraints. Instantiated with different objects, a general plan yields different action sequences or plans. Simple, single–action operators can often be combined into multiple–action general plans known as macro–operators. One way of achieving this is through the use of explanation–based learning.

4.1.1 Explanation–Based Learning

Explanation–based learning is a knowledge–intensive learning paradigm requiring a rich domain theory to construct the proofs or causal explanations enabling the learner to learn from single examples [DeJong86, Mitchell86]. Explanation–based learning involves two major steps: explanation and generalization. In the explanation step, an explanation is constructed for why the training example is an example of the goal concept. In the generalization step, the whole or parts of the explanation are generalized to come up with a general definition of the goal concept or more general subconcepts. Explanation and generalization can be carried out over situations and actions as well as control decisions. Here, we are concerned primarily with generalization over operators to create macro–operators or general plans, which are applicable over a wider class of situations than the specific plans derived from examples and usually lead to more efficient planning than planning with simple operators. In either case, by allowing concepts to be combined into more general concepts, learning in planning enables a planner to solve future instances of a learned general concept.

4.1.2 Problems with Standard EBL

Standard EBL would fail in situations better handled reactively for much the same reasons classical planning would. After execution, an explanation may be constructed which relies on data which became available only during execution. Thus, the resulting generalization would be useless in future situations where such information would be unavailable during plan construction.

For example, imagine the problem of learning how to cross the street. Consider a particular training example in which Mr. Cross, a street–crossing expert, approaches Mathews Ave., allows a green Chevy Nova, a white Datsun pick–up truck, and a gray Honda Accord to pass by, and finally walks across the street in between a red Toyota Tercel and a gold Camaro. Explaining this example would involve having to explain why it was that Mr. Cross was able to safely get to the other side of Mathews. This would include having to prove some type of *clear–path* precondition, which is satisfied in this particular example by the fact that there were two cars, the red Tercel and the gold Camaro, between which was a gap large enough to allow safe passage. The generalization would thus yield a plan whose preconditions include determining two such cars between which there exists a suitably–sized gap. Unfortunately, in future street–crossing instances, the information needed to satisfy such preconditions will be unavailable. Determining which two cars to cross between could potentially involve having to know the location of and path to be traveled by every car in the world to prove that they will not come in the way during crossing.

What ought to be learned is a general plan wherein the world is monitored during execution in order to dynamically determine which two cars to cross between. As discussed in the previous chapter, achievability proofs guaranteeing the existence of two such cars may be used to construct provably–correct plans similar to those of classical planning. Possible achievability proofs are that the street is closed–off due to construction, or that it is a lightly–traveled one–way street, or that it is past rush hour in the morning, or that there is a pedestrian light. Unlike the useless preconditions which may be extracted from a standard explanation such as that outlined previously, the preconditions extracted from these achievability proofs rely only on information available a priori and will thus yield plans which can be used in future street–crossing problems.

4.2 Contingent Explanation–Based Learning

Contingent EBL enables learning the type of plans exemplified by the street–crossing example presented in the previous section. These plans differ from ordinary plans in that they may involve deferring some goals until execution. Contingent EBL enables a learning system to reason about deferred goals through conjectured variables, which enables the construction of contingent explanations whose generalizations yield reactive plans.

4.2.1 The Learning Task

The integrated approach to planning presented in a previous chapter involves two phases of plan construction. Prior to execution, a set of partial plans from which only incomplete plans may be derived is constructed, with the constraint that all deferred goals be achievable. During execution, additional information may then be utilized in addressing the deferred goals and completing a particular plan. Such a plan which is incomplete prior to execution and completed during execution is called a *reactive plan.*

In pure reactive planning, every action is determined during execution and thus the reactive planner begins the execution phase with a trivial reactive plan composed of zero actions. On the other hand, in classical planning, plans are constructed completely a priori and thus only non–reactive plans are used. In the integrated approach, an incomplete reactive plan is constructed prior to execution, and a *completion* determined during execution. Since the incomplete reactive plan has only *achievable* missing components or deferred goals, all of which are satisfied during execution, it is a *completable* reactive plan.

Here we deal with the problem of learning general completable reactive plans. That is, the system's learning task is to learn general plans which can be used in the integrated approach for creating completable reactive plans and their associated completors.

4.2.2 Differentiating Achievable Deferred Goals from Unsatisfiable Planning Goals

In order to learn general completable reactive plans, deferred goals must be differentiated from unsatisfiable goals. During the observation of a training example, a learning system must be able to reason out when a particular planning goal can be deferred with good reason. It must realize when a particular goal was achievable and thus deferrable as opposed to when a particular goal was not a priori determined to be achievable but was achieved by chance. The system must also later on be able to differentiate between planning failures and deferred goals in its own planning.

Every general plan has associated with it the combined preconditions and effects of its operators. A planning failure results from either the inability to find some general plan with the desired effects, or the failure to satisfy the preconditions of any general plan with the desired effects. A completely–instantiated plan signifies success. However, a non–reactive plan may also sometimes contain incompletely–specified preconditions and actions. These cases occur if the domain knowledge and initial knowledge support the conclusion that some predicate is universally true or that some action can be applied to any object to achieve the desired result.[1] The uninstantiated variables in these incompletely–instantiated plans may be arbitrarily resolved and thus also signify success.

In the case of reactive plans, certain actions and certain preconditions may also intentionally be left incompletely–determined until execution time. Unlike values which may be arbitrarily chosen for the uninstantiated universally–quantified variables in non–reactive plans prior to execution, however, the values for the unspecified plan parameters in reactive plans may only be determined during execution—when a particular condition becomes true, after a certain action is executed, when certain properties can be precisely determined, etc. Unlike a planning failure, the eventual instantiation of these incompletely–determined plan components is guaranteed by the achievability proofs, much as the existence of a value for the uninstantiated universally–quantified variables in a non–reactive plan are guaranteed by proofs of correctness for the plan. A learning system must thus be able to differentiate between achievable deferred goals and unsatisfiable plan parameters.

1. Probably a rather bizarre but nevertheless theoretically possible idea.

4.2.2.1 Conjectured Variables

The notion of a conjectured variable is introduced as a means by which deferred goals can be differentiated from planning failures. A *conjectured variable* is a planner–posed existential used in place of an actual parameter value. A conjectured variable acts as a place–holder for the eventual values of those particular plan parameters whose exact instantiations will not be made until execution. In the hammering example presented earlier, the conjectured variable would be the number of times the pound action would be carried out. In the coffee example, the conjectured variable would be the cup. In both of these cases, the exact value eventually taken on by the plan parameters—the number of blows and exactly which cup—is not determined until the execution of the plan. Conjectured variables are a similar idea to the *formal objects* used in NOAH [Sacerdoti77] to act as place–holders for variable values.

In order to guarantee the success of a plan even with the presence of conjectured variables, we impose the restriction that a conjectured variable exist in place of an actual plan parameter value only if an associated achievability proof can be constructed. A conjectured variable used in this way is called a *valid* conjectured variable. Depending upon the domain knowledge representation in a particular implementation, a conjectured variable may be used in place of different types of values. It could be a time point in an interval–based representation, an upper bound on the iteration control of an operator, an object upon which an action is performed, or any of a number of other possible plan parameters.

For example, consider the following Intermediate Value Rule Schema from the previous chapter:

Intermediate Value Rule Schema

$$
\begin{array}{l}
[\forall\ \theta \\
\quad [\forall\ q\ v0\ v2\ t0\ t2 \\
\qquad (\quad (\text{value } q\ v0\ t0) \text{ AND} \\
\qquad\quad (\text{continuous } q) \text{ AND} \\
\qquad\quad ((\theta\ (t0\ t2)) \rightarrow (\text{value } q\ v2\ t2))\ ) \\
\quad \rightarrow \\
\qquad [\forall\ v1 \\
\qquad\quad ((\text{between } v1\ v0\ v2) \\
\qquad\quad \rightarrow\ [\exists\ t1\ ((\text{within } t1\ (t0\ t2)) \text{ AND } ((\theta\ (t0\ t1)) \rightarrow (\text{value } q\ v1\ t1)))]\ )\ ]\ ]\ ].
\end{array}
$$

Here the time point t1 is a conjectured variable. For any goal value v1, there is guaranteed to be a time point t1 within a particular interval (t0 t2) at which q will have the value v1 if the preconditions are satisfied. That is, the leaves of the proof tree whose root is the consequent of this rule constitute the *achievability conditions* which determine the achievability of a value for t1. Provided they can be satisfied, t1 is a valid conjectured variable, taking the place of the precise time point at which q reaches the value v1 during execution.

Earlier, it was pointed out that one of the problems with using standard EBL in reactive situations was that the generalized explanation could yield a general plan with preconditions which could not be satisfied a priori. In the contingent explanation–based learning approach presented here, these preconditions are replaced with achievability conditions. Thus in order for the plans supported by these achievability proofs to fare better than the plans resulting from standard EBL, the achievability conditions must be conditions which can be satisfied a priori. That is, they must reason only about knowledge available prior to execution.

4.2.2.2 Conjectured Variables vs. Uninstantiated Universally–Quantified Variables

As mentioned earlier, the actions of a non–reactive plan such as that constructed by a classical planner need not be completely determined for a plan to be considered applicable at plan construction time. That is, there may be operators of the form (OP ?obj object–931), where ?obj is an uninstantiated universally–quantified variable. Since the plans constructed by a classical planner are provably correct, the existence of such a variables means that it may be instantiated with any object in the world to yield a good plan. If such variables are left uninstantiated until just prior to the execution of the operator they exist in, they begin to seem very much like conjectured variables. In fact, a conjectured variable may be introduced by a planner to act as a place–holder for the eventual value of such an uninstantiated universally–quantified variable. Such a conjectured variable is also a valid conjectured variable since if the properties associated with the universally–quantified variable have been proven true of all the objects in the world, then it is trivially the case that there exists some object which has the desired properties.[2]

However, a planner need not wait until execution to determine values for these uninstantiated universally–quantified variables. Thus the use of conjectured variables in place of their eventual values is an arbitrary and unnecessary step. A more interesting use of conjectured variables is for values which are difficult to determine prior to execution but can be simply computed using execution–time information. For example, determining precisely how long to whip cream until soft peaks form would be a difficult a priori computation which would have to take into account the amount of cream, the temperature of the bowl, the speed of the beaters, the humidity in the kitchen, and so on. However, it is fairly simple during execution to monitor the consistency of the cream and decide to stop the beating process when soft peaks form. If it can be proven prior to execution that the goal whipped state is achievable, then a valid conjectured variable may act as a placeholder for the precise time at which the beaters must be stopped—a value to be determined during execution.

2. Assuming a non–empty world.

4.2.2.3 Conjectured Variables vs. Intervals and Situations

Another data type must be differentiated from conjectured variables—the data type for representing intervals in an interval-based representation or situations in a situation-based representation. Consider the following Terminal Value Rule Schema from the previous chapter:

Terminal Value Rule Schema

```
[∀ α
    [∀ q v1 v2 v3 v4 vg t1 t2 t3 t4
        (   (((value q v1 t1) AND (α 1 (t1 t2) AND (value q v2 t2)) AND (≠ v1 vg))
             → (between v2 v1 vg))
            AND
            (((value q v1 t1) AND   (α 1 (t1 t2) AND (value q v2 t2) AND (= v1 vg))
             → (= v2 vg))
            AND
            (NOT   (((value q v1 t1) AND   (α 1 (t1 t2) AND (value q v2 t2) AND
                     (value q va ta) AND (α 1 (ta tb)) AND (value q vb tb) AND
                     (after (t1 t2) (ta tb)) AND (< (– vb va) (– tg t1)))
                    →  (< (– v2 v1) (– vb va)) ) ) )
    →
        [∃ n tn  ((α n (t1 tn) → (value q vg tn))] ] ].
```

This rule schema involves the conjectured variable n, which is the number of times the action or action sequence α will be executed to achieve the goal value vg. There is, however, also the existentially-quantified variable tn, which seems very much like the conjectured variable n. It seems to have been proposed by the planner to take the place of the eventual value of the end of the interval (t1 tn).

It is true that like a conjectured variable, tn is meant to act as a place-holder—in this case, for the time point specifying the end of the interval (t1 tn) which is the interval associated with performing the action sequence α(n). Unlike a conjectured variable, however, interval endpoints such as tn have their value determined by the action or qualitative state they are associated with. That is, tn exists simply because the interval over which α(n) occurs must be named. In interval-based formulations, intervals are usually represented using a separate data type, thus explicitly recognizing this difference. The same is true in the implementation used here, where &-prefaced names are used for interval endpoints such as tn (i.e. &tn) in order to differentiate them from !-prefaced conjectured variables such as n (i.e. !n). In situation-based representations, the interval-naming problem manifests itself as a situation-naming problem, with situations, like intervals, needing names. Again, one solution is to adopt a similar naming convention allowing situation names to be differentiated from conjectured variables.

4.2.3 Modified EBL

The explanations constructed and generalized in classical planning involve planning problems wherein an a priori explanation is exactly the same as an explanation constructed after execution. In the integrated approach, however, an explanation constructed after execution may not be the same as an a priori explanation, due to the additional information which becomes available during execution and is used to complete the plan. This difference is captured with conjectured variables. The only difference in the explanations of a reactive plan constructed before and after execution is that before execution, the conjectured variables will not be bound whereas after execution they will all have been bound. Thus the main difference between explanations of reactive plans and non-reactive plans is the presence of conjectured variables in the explanations of reactive plans.

A *contingent explanation* is an explanation which involves the use of conjectured variables. The validity of a such an explanation is contingent upon the existence of actual parameter values to replace conjectured variables during execution—hence the name. Provided all the conjectured variables in a contingent explanation are valid conjectured variables—i.e. conjectured variables for which associated achievability proofs are constructed—the explanation incorporates the achievability conditions of the deferred goals. Since the achievability conditions are satisfiable with knowledge available prior to execution, the generalization of a contingent explanation having only valid conjectured variables will yield a general plan which may be used in future instances.

4.3 Planning with General Completable Plans

We have discussed how reactive situations may be understood in terms of contingent explanations which incorporate conjectured variables. These contingent explanations, when generalized, yield general plans differing from those of standard explanation-based learning only in that they contain conjectured variables. These general plans thus enable a planner to construct reactive plans in which conjectured variables are used in place of actual parameter values. During plan construction, conjectured variables are simply treated as constants. During execution, however, appropriate values must be found to replace these conjectured variables in order to complete the plan.

The term *completor* will be used to denote the procedure by which an appropriate value is determined for a conjectured variable. The completors discussed here serve the purpose of constructing completions to the completable reactive plans constructed in the integrated approach. A completor to determine an appropriate value must be constructed for every conjectured variable. In the previous chapter, three types of achievability proofs were discussed—specifically proofs for the achievability of terminal values, intermediate values, and one-of-many. Each one is associated with a particular completor (Table 4-1), which will now be discussed in turn.

4.3.1 Iterators

The first type of achievability proof discussed was for terminal values. This involves a conjectured variable n which specifies how many times a particular action or action sequence α should be performed in order to achieve the goal. The corresponding completor for a conjectured variable of this type is an *iterator*, which is a looping construct by which an action can be repeatedly executed until certain conditions—specifically the goal conditions—are met. In this case, the goal condition is that a particular quantity q have the value vg. Thus the iterator simply checks at the end of each execution of α whether q has the value vg. If so, then α has been executed enough times, i.e. n times, and the repetitions of α can be halted.

CONJECTURED VARIABLE	COMPLETOR
terminal value repeated action; incremental progress towards goal until goal is reached	*iterator* repeat action until goal is reached
intermediate value continuous monotonic process; goal quantity value between initial and projected extreme	*monitor* observe changing quantity until goal value is reached
one–of–many many objects of the same type	*filter* choose appropriate object

Table 4–1. Conjectured Variables and Corresponding Completors.

4.3.2 Monitors

The second type of achievability proof was for intermediate values of a continuously–changing quantity. This involves a conjectured variable t1 which represents the time point at which a particular quantity q reaches its goal value v1. The corresponding completor for this type of conjectured variable is a *monitor*, which involves monitoring the world for certain conditions and signaling when such conditions are met. In this case, the monitor must observe the changing quantity q until it reaches the value v1, at which point time t1 has been reached. Some action in the plan which is to be executed at time t1 can now be executed.

4.3.3 Filters

The third type of achievability proof involved proving the existence of one from the existence of many. This involves a conjectured variable obj representing the precise object of type τ upon which an action a may be executed in order to achieve the goal. The corresponding completor in this case is a *filter*, which serves to find an object of type τ during execution. That is, the filter analyzes objects for type until it finds one which is of type τ, which then becomes the value of obj.

4.3.4 Others

At this point, the types of reactive planning tasks which have been addressed have needed only simple plan completions, which are easily determined during execution. Thus the problem of constructing completors has not been thoroughly addressed. Completors serve the purpose of determining completions for reactive plans, and depending upon the kind of value sought for a particular conjectured variable, the actions associated with completors may involve observing the environment, measuring quantities, performing mathematical computations, determining the truth of certain propositions, testing for a desired condition, and so on. The three types presented here are just three possible types of completors. Finding a general solution for constructing plan completions from the right or various kinds of completors later on is an interesting area for future work.

Chapter 5
IMPLEMENTATION AND EMPIRICAL COMPARISONS

Imagine the problem of docking a spaceship. This involves maneuvering the spaceship towards the landing dock and slowing it down so that it stops in its berth (Figure 5–1). In order to ensure a successful docking,

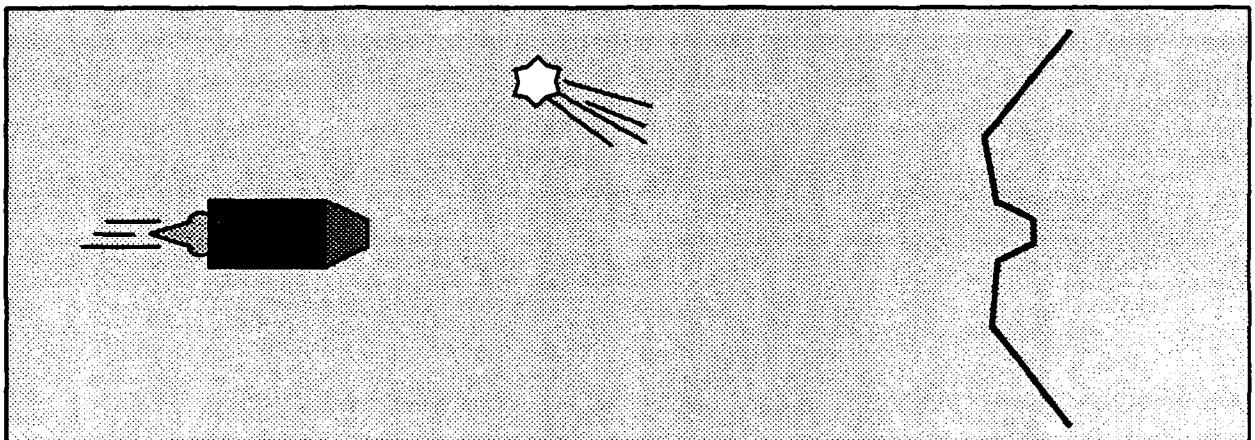


Figure 5–1. Docking a spaceship.

the spaceship must be decelerated at a rate such that at the time it reaches the landing dock, it comes to a full stop or is traveling at a speed no more than a certain very small amount which can be "absorbed" by the landing dock. Under ideal conditions, a classical planner could accurately determine the deceleration rate by simply plugging the appropriate values into equations derived from principles taught in introductory physics. In the real world, however, countless factors must be considered in the computation of the deceleration rate—such as fuel flow, energy output, heat loss, engine efficiency, and friction, which in turn entail the consideration of more factors such as possible blockages in the fuel line, efficiency of combustion, temperature differentials, material characteristics, nozzle geometry, and so on. Failure to account for all these will likely result in a computed deceleration rate which varies from the actual deceleration rate achieved during execution. In many cases, the difference may not matter, but in others, it may mean the difference between a successful and an unsuccessful docking.

The basic problem lies in the insistence on complete a priori computation. If, instead, a system were allowed to dynamically determine certain deceleration rates, velocities, and/or distances during execution, rather than compute all of these before execution, then it can be made sensitive to the execution environment. Certain runtime variations may be very easy to detect and account for during execution, but very computationally expensive to predict a priori. Problems like this are prime candidates for the integrated approach to planning.

In this chapter, the performance of an implemented system on an acceleration example is discussed. First, a brief overview of the system is given. Then, the explanation–based learning of a reactive acceleration plan by the system is presented. Included are a discussion of the explanations constructed throughout the example and the generalization procedure which yields a generalized reactive plan with completing components. Finally, an evaluation of reactive vs. non-reactive plans, including a comparison of the performance of a reactive plan vs. a non-reactive plan on some hypothetical acceleration problems, is presented.

5.1 Implementation Details

The system is written in Common LISP and runs on an IBM RT Model 125. The explanation-based learning module of the system uses a modified EGGS [Mooney86] algorithm in learning generalized plans. Currently, the planning module is able to plan only by instantiating the generalized plans in its plan library. The system uses the interval-based representation scheme discussed in Chapter 4, which allows predications over time points as well as time intervals. In the rules and explanations presented below, universal variables are prefaced with "?", conjectured variables with "!", and interval endpoints with "&".

5.2 Example

The docking problem presented above involves the two interdependent tasks of deceleration and motion. The system was run on a simpler version involving only the problem of acceleration, where the task is to learn how to achieve a particular greater velocity from some initial velocity by analyzing an example of such an acceleration. The learning scenario involves an expert providing an example, for which the system then constructs an explanation, which it generalizes to form a generalized plan in the manner discussed in the previous chapter. During learning, the system has access to the same initial and execution-time information used by the expert, but it has to reason on its own about how the reactive plan to achieve the goal was constructed prior to and completed during execution.

5.2.1 Initial State

Initially, at time point 10.0, the spaceship Shipshape is moving at a velocity of 65.0 m/s and its rockets are off. These conditions are represented by the following propositions:

(value (velocity Shipshape) 65.0 10.0)

(off_rockets Shipshape (-100.0 &t0)).

5.2.2 Goal

The goal is to have Shipshape moving at a velocity of 100.0 m/s at some time in the future—i.e. after the current time point 10.0. This is represented by the following propositions:

(moving Shipshape 100.0 (&ts &tg))

(after &ts 10.0)

(after &tg 10.0).

5.2.3 Observed Events

The plan which the expert uses to achieve the goal consists of two actions. After the execution of each action, the system explains how the action was executable by determining how its preconditions are satisfied at the time the action was executed. The first action is that of firing the rockets at time point 10.0:

(fire-rockets Shipshape 10.0).

The second action is that of stopping the rockets at time point 35.0:

(stop-fire-rockets Shipshape 35.0).

The system observes that prior to the fire-rockets action, the spaceship was travelling at 65.0 m/s. As the rockets were firing, its velocity increased. When the rockets stopped firing, it was travelling at 100.0 m/s, a velocity which it maintained from then on. The fire–rockets action has a precondition that the spaceship's rockets be off at the time of firing. The system confirms that this is satisfied by an initial condition. The stop–fire–rockets action has a precondition that the spaceship's rockets be on at the time of firing. The system confirms that this is satisfied by an effect of the previous fire–rockets operator.

5.3 Explanation

The system confirms that the two actions of firing the rockets and stopping them achieves the goal and then explains how the goal was achieved by constructing an explanation for how the observed plan resulted in the spaceship moving at 100.0 m/s over some future time interval. The explanation the system constructs is shown in Figure 5–2.

The goal is to have the spaceship moving at 100.0 m/s over some future time interval to be designated by (&ts &tg).[1] This goal is satisfied because the spaceship was moving at 100.0 m/s at time point 17.1576, and it was moving at a constant rate over the open–ended time interval (17.1576 &t24036). Shipshape reached a velocity of 100.0 m/s because at time 10.0 it had the lower velocity of 65.0 m/s and its velocity was increasing over the time interval (10.0 17.1576). This sub–explanation may not strike one as a valid explanation. After all, just because something is increasing doesn't mean it will be a specific value at a specific time. However, this sub–explanation results from the instantiation of the following Intermediate Value Rule derived from the Intermediate Value Rule Schema of Chapter 4:[2]

```
Intermediate Value Rule
    (   (value ?q ?v0 ?t0) AND
        (continuous ?q) AND
        ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) AND
        (between ?v1 ?v0 ?v2) AND
        (qualitative_behavior ?q ?beh (?t0 !t1) )
    →
    (   (value ?q ?v1 !t1) ).
```

Using this rule, the system can reason that since the initial velocity is 65.0 and that the maximum velocity of 500.0 can be reached if the velocity is made to increase till the end of time, then there exists some time before then and after the current time such that if the velocity increases until then, it will have the intermediate goal value of 100.0. In this particular example, that time point turned out to be time 17.1576. Finally, the constant velocity over the interval is explained by the fact that the rockets are off over that time, due to the stop–fire–rockets action at the start of that interval.[3]

1. &ts and &tg are unbound interval endpoint variables constrained only to be after time point 17.0. The system also maintains other interval constraints implicitly—for example, for any interval (ta tb), tb must be on or after ta.

2. See Appendix A for a discussion on how FOPC rules are derived from rule schemata.

3. In this example, an explicit assumption is made that the sum of the external forces upon the spaceship is negligible—i.e. the only force which may act upon the spaceship comes from the firing of its rockets.

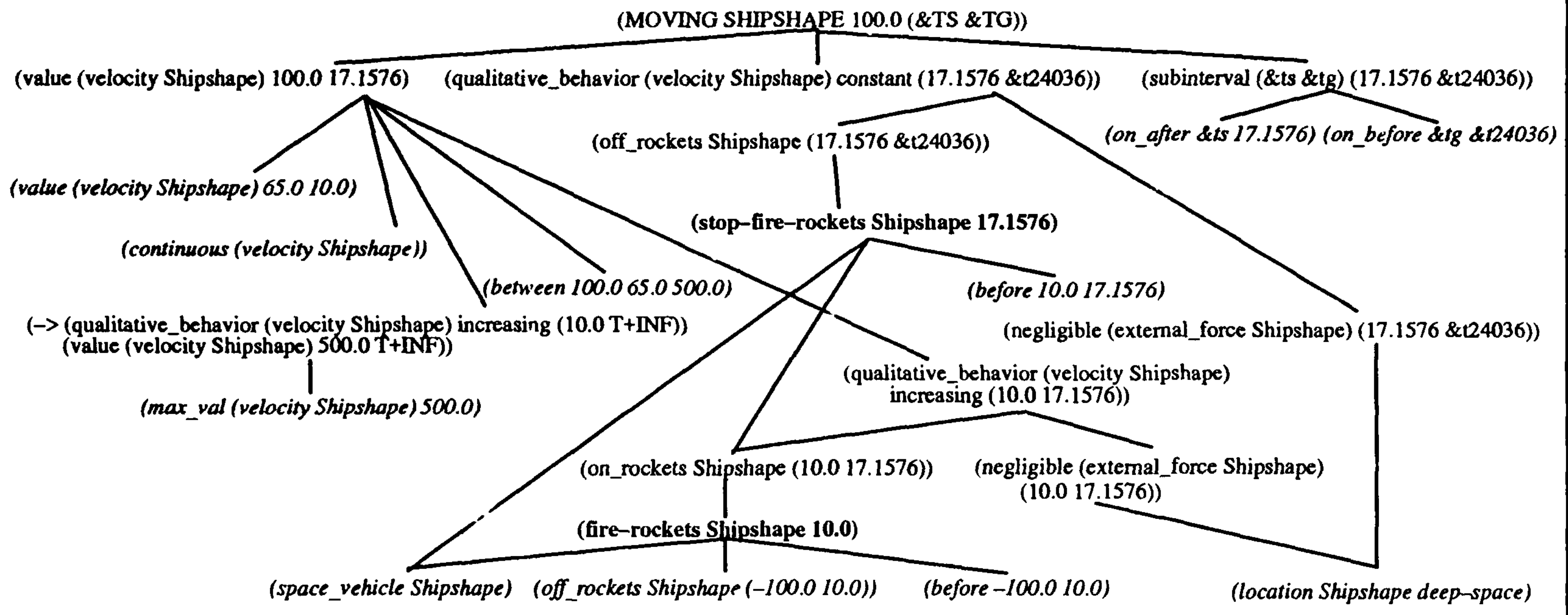

(MOVING SHIPSHAPE 100.0 (&TS &TG))
(value (velocity Shipshape) 100.0 17.1576)
(qualitative_behavior (velocity Shipshape) constant (17.1576 &t24036))
(subinterval (&ts &tg) (17.1576 &t24036))
(off_rockets Shipshape (17.1576 &t24036))
(on_after &ts 17.1576)
(on_before &tg &t24036)
(value (velocity Shipshape) 65.0 10.0)
(stop-fire-rockets Shipshape 17.1576)
(continuous (velocity Shipshape))
(between 100.0 65.0 500.0)
(before 10.0 17.1576)
(-> (qualitative_behavior (velocity Shipshape) increasing (10.0 T+INF))
(value (velocity Shipshape) 500.0 T+INF))
(negligible (external_force Shipshape) (17.1576 &t24036))
(qualitative_behavior (velocity Shipshape) increasing (10.0 17.1576))
(max_val (velocity Shipshape) 500.0)
(on_rockets Shipshape (10.0 17.1576))
(negligible (external_force Shipshape) (10.0 17.1576))
(fire-rockets Shipshape 10.0)
(space_vehicle Shipshape)
(off_rockets Shipshape (-100.0 10.0))
(before -100.0 10.0)
(location Shipshape deep-space)

Figure 5-2. Specific explanation constructed in integrated approach.

5.4 Generalization

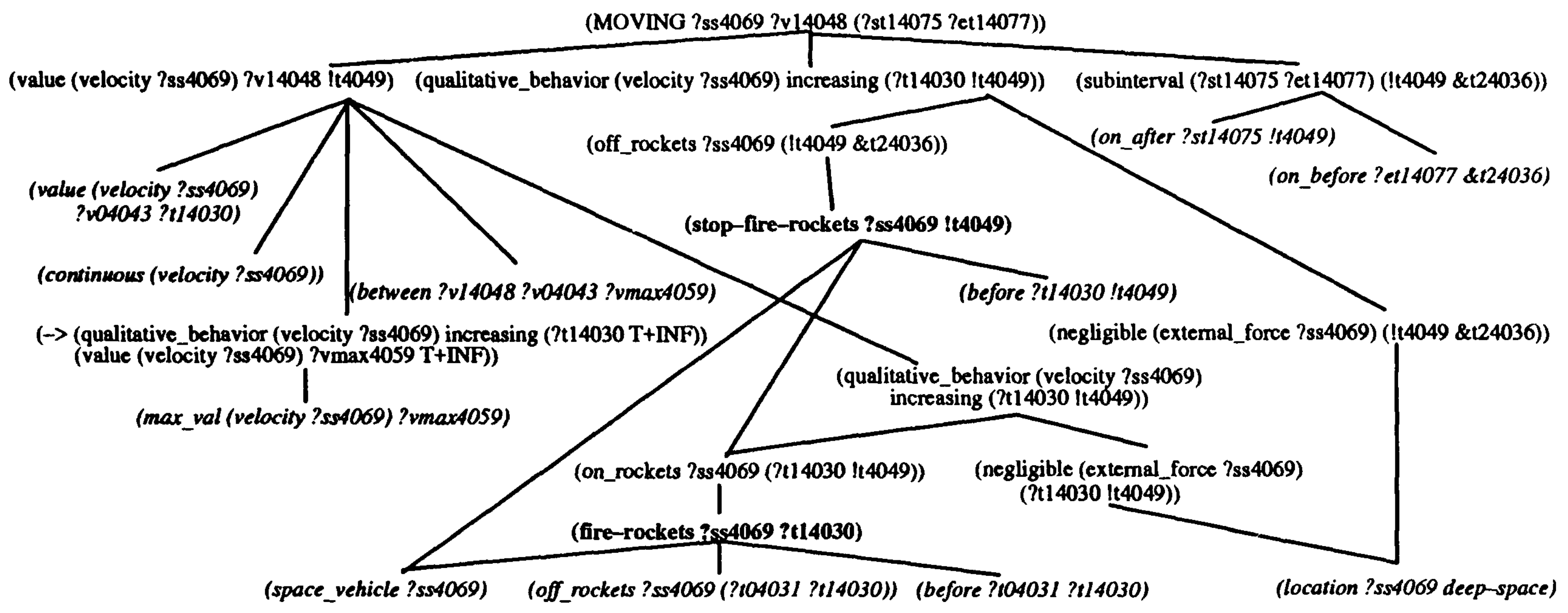
Using the modified generalization algorithm discussed in the previous chapter, the system processes the explanation to yield the generalized explanation shown in Figure 5–3, from which is derived the following general plan:

```
[   GOAL
        (moving ?ss4069 ?v14048 (?st14075 ?et14077))
    PRECONDITIONS
        (value (velocity ?ss4069) ?v04043 ?t14030)
        (continuous (velocity ?ss4069))
        (max_val (velocity ?ss4069) ?vmax4059)
        (between ?v14048 ?v04043 ?vmax4059)
        (space_vehicle ?ss4069)
        (off_rockets ?ss4069 (?t04031 ?t14030))
        (before ?t04031 ?t14030)
        (before ?t14030 !t4049)
        (location ?ss4069 deep–space)
        (on_after ?st14075 !t4049)
        (on_before ?et14077 &t24036)
    OPERATORS
        (fire–rockets ?ss4069 ?t14030)
        (stop–fire–rockets ?ss4069 !t4049)   ].
```

Now suppose that this plan is used in planning for some other acceleration problem. The resulting plan will have the fire–rockets action being performed at that time ?t14030 which is the initial given time, while the stop–fire–rockets action will be performed at some unbound time !t4049. One more step must be performed before this plan can be used to yield a completable reactive plan. A completor must be constructed and incorporated as an additional operator in the general plan to find an appropriate value for !t4049 during execution.

As discussed in the previous chapter, completors are constructed by analyzing the generalized explanation as to the conditions surrounding the conjectured variables. First, all the *non–operational* components in the general plan are identified. A plan precondition, effect, or operator is considered *operational* if values can be found for all its variables. Since aconjectured variable does not have a value prior to execution, a completor must be constructed to render it operational. The general plan above contains the following non–operational components:

```
preconditions
    (before ?t14030 !t4049)
    (on_after ?st14075 !t4049)
operators
    (stop–fire–rockets ?ss4069 !t4049).
```



(MOVING ?ss4069 ?v14048 (?st14075 ?et14077))
(value (velocity ?ss4069) ?v14048 !t4049)
(qualitative_behavior (velocity ?ss4069) increasing (?t14030 !t4049))
(subinterval (?st14075 ?et14077) (!t4049 &t24036))
(on_after ?st14075 !t4049)
(on_before ?et14077 &t24036)
(off_rockets ?ss4069 (!t4049 &t24036))
(value (velocity ?ss4069) ?v04043 ?t14030)
(stop-fire-rockets ?ss4069 !t4049)
(continuous (velocity ?ss4069))
(between ?v14048 ?v04043 ?vmax4059)
(before ?t14030 !t4049)
(-> (qualitative_behavior (velocity ?ss4069) increasing (?t14030 T+INF)) (value (velocity ?ss4069) ?vmax4059 T+INF))
(negligible (external_force ?ss4069) (!t4049 &t24036))
(qualitative_behavior (velocity ?ss4069) increasing (?t14030 !t4049))
(max_val (velocity ?ss4069) ?vmax4059)
(on_rockets ?ss4069 (?t14030 !t4049))
(negligible (external_force ?ss4069) (?t14030 !t4049))
(fire-rockets ?ss4069 ?t14030)
(space_vehicle ?ss4069)
(off_rockets ?ss4069 (?t04031 ?t14030))
(before ?t04031 ?t14030)
(location ?ss4069 deep-space)

Figure 5-3. Generalized explanation in integrated approach.

These three components involve the conjectured variable !t4049, thus necessitating a completor. From the generalized explanation, it is determined that this conjectured variable is introduced by the following proposition:

(value (velocity ?ss4069) ?v14048 !t4049).

A monitor is thus created for !t4049 using this condition. This operator determines a value for !t4049 during execution by monitoring the observable quantity of velocity until the goal value of ?v14048 is reached. The time at which this is achieved is exactly the time point to which !t4049 needs to be bound.

The final general plan is thus:

```
[  GOAL
       (moving ?ss4069 ?v14048 (?st14075 ?et14077))
   PRECONDITIONS
       (value (velocity ?ss4069) ?v04043 ?t14030)
       (continuous (velocity ?ss4069))
       (max_val (velocity ?ss4069) ?vmax4059)
       (between ?v14048 ?v04043 ?vmax4059)
       (space_vehicle ?ss4069)
       (off_rockets ?ss4069 (?t04031 ?t14030))
       (before ?t04031 ?t14030)
       (before ?t14030 !t4049)
       (location ?ss4069 deep-space)
       (on_after ?st14075 !t4049)
       (on_before ?et14077 &t24036)
       (observable (velocity ?ss4069))
   OPERATORS
       (fire-rockets ?ss4069 ?t14030)
       (stop-fire-rockets ?ss4069 !t4049)
       [MONITOR for (value (velocity ?ss4069) ?v14048 !t4049)] ].
```

5.5 Non-reactive Plan vs. Reactive Plan

The system was also run on the acceleration example in a classical planning mode, where it constructs a standard explanation involving no conjectured variables. The system explained the time at which the rocket firing is stopped to be a precise time point computed using equations derived from the principle of the conservation of linear momentum.[4] The explanation, shown in Figure 5–4, is generalized (Figure 5–5) and used to construct the following general non-reactive plan:

4. See Appendix B for the solution of this physics problem.

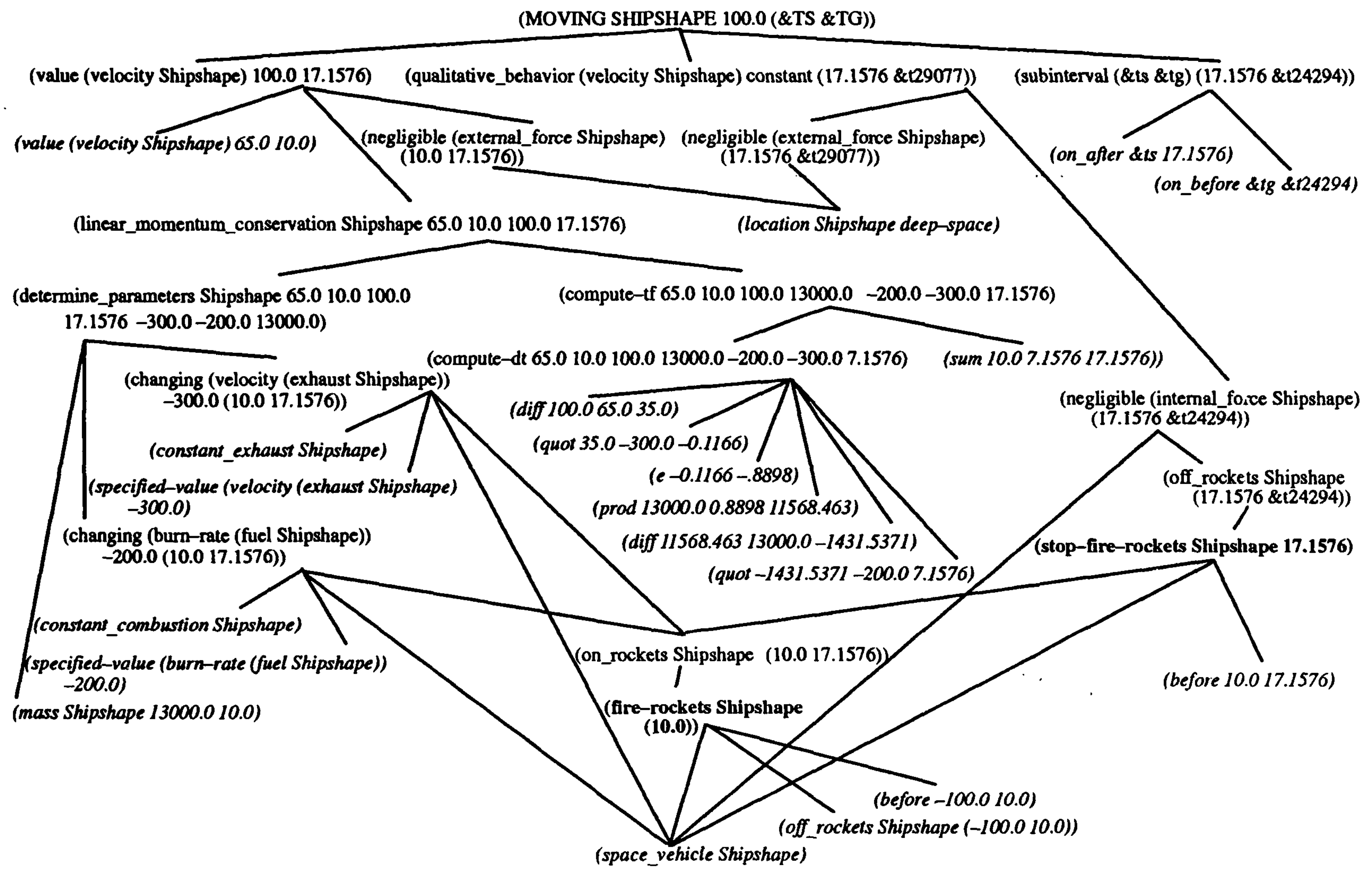

(MOVING SHIPSHAPE 100.0 (&TS &TG))
(value (velocity Shipshape) 100.0 17.1576)
(qualitative_behavior (velocity Shipshape) constant (17.1576 &t29077))
(subinterval (&ts &tg) (17.1576 &t24294))
(value (velocity Shipshape) 65.0 10.0)
(negligible (external_force Shipshape) (10.0 17.1576))
(negligible (external_force Shipshape) (17.1576 &t29077))
(on_after &ts 17.1576)
(on_before &tg &t24294)
(linear_momentum_conservation Shipshape 65.0 10.0 100.0 17.1576)
(location Shipshape deep-space)
(determine_parameters Shipshape 65.0 10.0 100.0 17.1576 -300.0 -200.0 13000.0)
(compute-tf 65.0 10.0 100.0 13000.0 -200.0 -300.0 17.1576)
(compute-dt 65.0 10.0 100.0 13000.0 -200.0 -300.0 7.1576)
(sum 10.0 7.1576 17.1576))
(changing (velocity (exhaust Shipshape)) -300.0 (10.0 17.1576))
(diff 100.0 65.0 35.0)
(quot 35.0 -300.0 -0.1166)
(e -0.1166 -.8898)
(prod 13000.0 0.8898 11568.463)
(diff 11568.463 13000.0 -1431.5371)
(quot -1431.5371 -200.0 7.1576)
(negligible (internal_force Shipshape) (17.1576 &t24294))
(constant_exhaust Shipshape)
(specified-value (velocity (exhaust Shipshape) -300.0)
(changing (burn-rate (fuel Shipshape)) -200.0 (10.0 17.1576))
(off_rockets Shipshape (17.1576 &t24294))
(stop-fire-rockets Shipshape 17.1576)
(constant_combustion Shipshape)
(specified-value (burn-rate (fuel Shipshape)) -200.0)
(mass Shipshape 13000.0 10.0)
(on_rockets Shipshape (10.0 17.1576))
(fire-rockets Shipshape (10.0))
(before 10.0 17.1576)
(before -100.0 10.0)
(off_rockets Shipshape (-100.0 10.0))
(space_vehicle Shipshape)

Figure 5–4. Specific explanation constructed in classical planning mode.

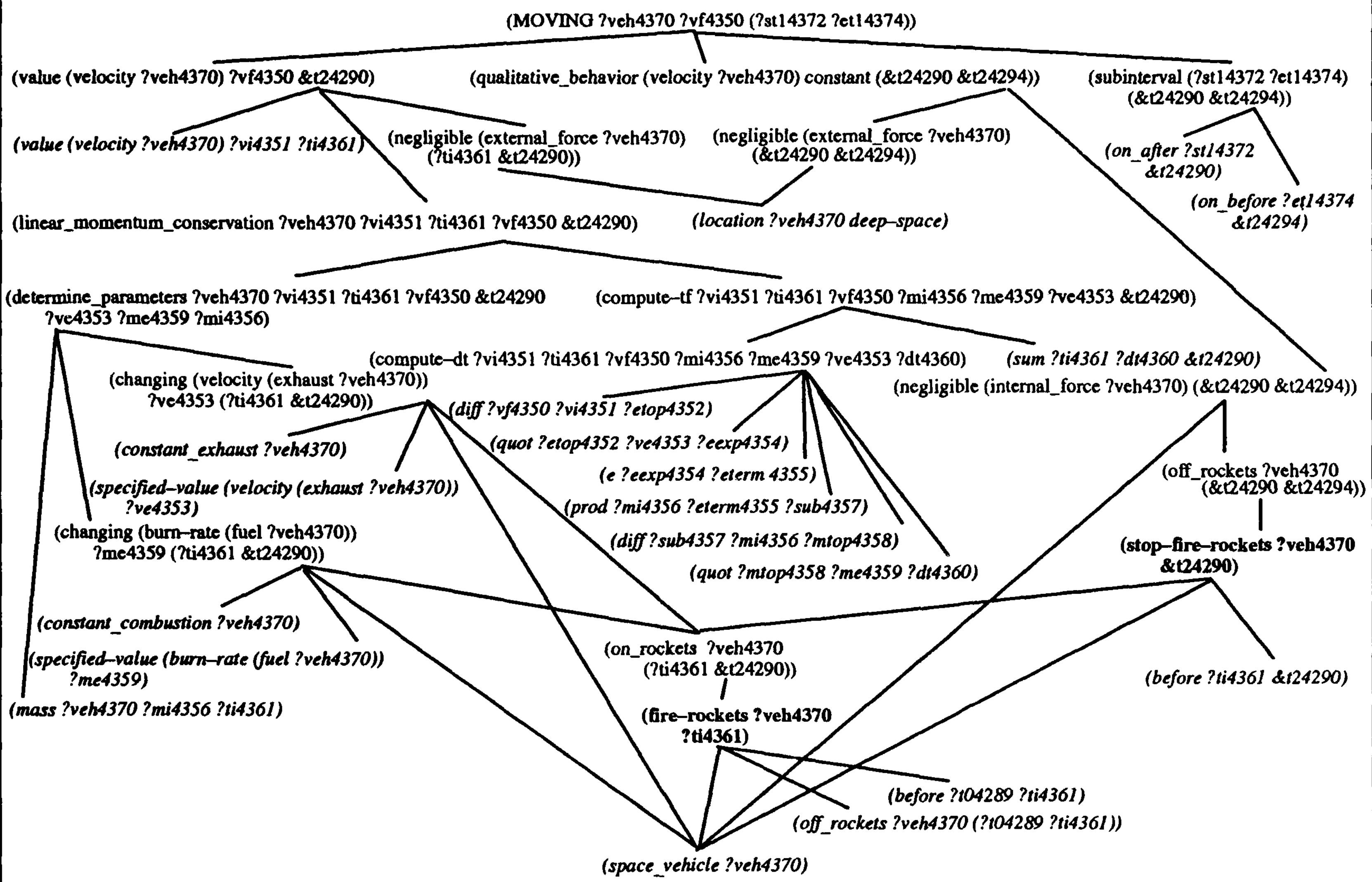

(MOVING ?veh4370 ?vf4350 (?st14372 ?et14374))
(value (velocity ?veh4370) ?vf4350 &t24290)
(qualitative_behavior (velocity ?veh4370) constant (&t24290 &t24294))
(subinterval (?st14372 ?et14374) (&t24290 &t24294))
(value (velocity ?veh4370) ?vi4351 ?ti4361)
(negligible (external_force ?veh4370) (?ti4361 &t24290))
(negligible (external_force ?veh4370) (&t24290 &t24294))
(on_after ?st14372 &t24290)
(on_before ?et14374 &t24294)
(linear_momentum_conservation ?veh4370 ?vi4351 ?ti4361 ?vf4350 &t24290)
(location ?veh4370 deep-space)
(determine_parameters ?veh4370 ?vi4351 ?ti4361 ?vf4350 &t24290 ?ve4353 ?me4359 ?mi4356)
(compute-tf ?vi4351 ?ti4361 ?vf4350 ?mi4356 ?me4359 ?ve4353 &t24290)
(compute-dt ?vi4351 ?ti4361 ?vf4350 ?mi4356 ?me4359 ?ve4353 ?dt4360)
(sum ?ti4361 ?dt4360 &t24290)
(changing (velocity (exhaust ?veh4370)) ?ve4353 (?ti4361 &t24290))
(negligible (internal_force ?veh4370) (&t24290 &t24294))
(diff ?vf4350 ?vi4351 ?etop4352)
(constant_exhaust ?veh4370)
(quot ?etop4352 ?ve4353 ?eexp4354)
(e ?eexp4354 ?eterm 4355)
(specified-value (velocity (exhaust ?veh4370)) ?ve4353)
(prod ?mi4356 ?eterm4355 ?sub4357)
(off_rockets ?veh4370 (&t24290 &t24294))
(changing (burn-rate (fuel ?veh4370)) ?me4359 (?ti4361 &t24290))
(diff ?sub4357 ?mi4356 ?mtop4358)
(stop-fire-rockets ?veh4370 &t24290)
(quot ?mtop4358 ?me4359 ?dt4360)
(constant_combustion ?veh4370)
(on_rockets ?veh4370 (?ti4361 &t24290))
(specified-value (burn-rate (fuel ?veh4370)) ?me4359)
(before ?ti4361 &t24290)
(mass ?veh4370 ?mi4356 ?ti4361)
(fire-rockets ?veh4370 ?ti4361)
(before ?t04289 ?ti4361)
(off_rockets ?veh4370 (?t04289 ?ti4361))
(space_vehicle ?veh4370)

Figure 5–5. Generalized explanation in classical planning mode.

```
[   PRECONDITIONS
        (value (velocity ?veh4370) ?vi4351 ?ti4361)
        (constant_exhaust ?veh4370)
        (specified-value (velocity (exhaust ?veh4370)) ?ve4353)
        (constant_combustion ?veh4370)
        (specified-value (burn-rate (fuel ?veh4370)) ?me4359)
        (mass ?veh4370 ?mi4356 ?ti4361)
        (diff ?vf4350 ?vi4351 ?etop4352)
        (quot ?etop4352 ?ve4353 ?eexp4354)
        (e ?eexp4354 ?eterm4355)
        (prod ?mi4356 ?eterm4355 ?sub4357)
        (diff ?sub4357 ?mi4356 ?mtop4358)
        (quot ?mtop4358 ?me4359 ?dt4360)
        (sum ?ti4361 ?dt4360 &t24290)
        (location ?veh4370 deep-space)
        (space_vehicle ?veh4370)
        (off_rockets ?veh4370 (?t04289 ?ti4361))
        (before ?t04289 ?ti4361)
        (before ?ti4361 &t24290)
        (on_after ?st14372 &t24290)
        (on_before ?et194374 &t24294)
    GOAL
        (moving ?veh4370 ?vf4350 (?st14372 ?et14374))
    OPERATORS
        (fire-rockets ?veh4370 ?ti4361)
        (stop-fire-rockets ?veh4370 &t24290)  ].
```

As with the general reactive plan constructed in the integrated planning mode, this general non-reactive plan consists of two operators—a fire-rockets operator and a stop-fire-rockets operator. However, whereas the time !t4049 at which the stop-fire-rockets action is determined during execution in the general reactive plan, the time &t24290 at which the stop-fire-rockets action is to be performed in the general non-reactive plan is computed a priori. The value of &t24290, which is the end of the interval starting at ?ti4361 over which the rockets are on, is computed in the precondition (sum ?ti4361 ?dt4360 &t24290)—i.e. &t24290 = ?ti4361 + ?dt4360.

Consider now the performance of the system using the general reactive acceleration plan and the general non-reactive acceleration plan. Take a hypothetical acceleration problem, where the spaceship to be accelerated is moving at an initial velocity of v_i at time t_i, and the goal is to have it moving at a velocity of v_f over some future time interval. The system might construct the following plan using the general reactive plan:

Reactive Plan

[**fire–rockets** at time t_i

monitor increasing velocity for value v_f, binding t_f to time v_f is reached

stop–fire–rockets at time t_f]

In this plan, the time t_f at which the stop–fire–rockets action is to be performed is determined during execution by monitoring the velocity for the goal value v_f. Using the general non–reactive plan, the system might construct the following plan:

Non–Reactive Plan

[**fire–rockets** at time t_i

wait for time t

$$\text{where } t = \frac{e^{\frac{v_f - v_i}{v_e}} M_i - M_i}{m_e}$$

given v_i = velocity at time t_i

v_f = goal velocity

v_e = exhaust velocity

m_e = burn rate

M = total mass of spaceship at time t_i

stop–fire–rockets at time $t_f = t_i + t$]

In this plan, the time t_f at which the stop–fire–rockets action is to be performed is computed a priori using an equation derived from the principle of the conservation of linear momentum. Both the reactive plan and non–reactive plan predict that the spaceship will be moving at the goal velocity v_f by the time the stop–fire–rockets action is carried out.

The determination of t_f in the non–reactive plan, however, relies on certain assumptions. In particular, a constant rate of fuel combustion and a constant exhaust velocity over the time of rocket firing are implicit in the system's knowledge about Shipshape. Provided these assumptions hold during execution, rocket firing will proceed as predicted and the non–reactive plan will achieve the same results as the reactive plan. In the real world, however, there are a great number of factors which may affect the precise rate at which a spaceship accelerates. Fuel type, fuel to oxygen ratio, fuel tank pressure, oxygen tank pressure, flame temperature, completeness of combustion, combustion chamber dimensions, and nozzle shape are just some of the factors a system would have to take into account to accurately predict the acceleration rate. Any of these factors alone or in combination with others may affect the precise rate at which fuel is burned or the velocity at which the exhaust gases leave the spaceship—and consequently, the acceleration rate.

To test the effects of variations in the actual burn rate and actual exhaust velocity from the expected burn rate and expected exhaust velocity, the system was run on a set of data points which varied the actual values up to a maximum of 5% from the expected values. In the test problem, the initial velocity v_i was 50.0 m/s at the initial time t_i of 70.0, and the desired goal velocity v_f was 100.0 m/s. The total mass of the spaceship was also known to be 20000 kg at time 70.0. The expected burn rate m_e was -200.0 kg/s, and the expected (relative) exhaust velocity v_e was -300 m/s. If the expected burn rate and expected exhaust velocity are accurate predictions of the actual burn rate and the actual velocity, at the time t_f of 85.3518 computed a priori by the non-reactive plan the spaceship will indeed be traveling at 100 m/s. However, deviations from the expected burn rate of fuel affect the resulting final velocity of the non-reactive plan as shown in Table 5-1. A deviation as small as

EFFECTS OF VARIATIONS IN BURN RATE (m_e)				
deviation of me_{actual} from $me_{expected}$	Reactive Plan		Non-Reactive Plan	
	t_f	v_f	t_f	v_f
0%	85.3518	100.0000	85.3518	100.0000
-0.5%	85.4290	100.0000	85.3518	99.7281
-1%	85.5069	100.0000	85.3518	99.4564
-2%	85.6651	100.0000	85.3518	98.9138
-5%	86.1598	100.0000	85.3518	97.2918
+0.5%	85.2754	100.0000	85.3518	100.2722
+1%	85.1998	100.0000	85.3518	100.5446
+2%	85.0508	100.0000	85.3518	101.0901
+5%	84.6208	100.0000	85.3518	102.7328

Table 5-1. Effects of variations in burn rate on final velocity reached in reactive plan and non-reactive plan.

half a percent results in the spaceship moving at a velocity over a quarter of a meter per second—or almost a kilometer per hour—more or less than the expected velocity. Similar effects result from small deviations in the expected exhaust velocity (Table 5-2). When both the actual burn rate and actual exhaust velocity differ from their expected values, the effects on the final velocity reached in the non-reactive plan are even larger (Table 5-3).

In contrast, the final velocity reached in the reactive plan is always the goal of 100.0 m/s, since the value of t_f is determined during execution by monitoring the increasing velocity for this specific value. Small deviations from the expected burn rate and expected exhaust velocity do not affect the performance of the system using the reactive plan.

Achieving a final velocity that is a mere fraction of a meter per second or even a few meters per second off the goal velocity may be acceptable in isolated acceleration problems. However, in larger planning problems where acceleration to a particular velocity is but one of many similarly-constrained problems, the propagation of these small errors may result in larger and larger errors which eventually cause a plan to fail. The non-reactive plan, with its reliance on pre-computed values, is thus much more susceptible to failure than the reac-

EFFECTS OF VARIATIONS IN EXHAUST VELOCITY (v_e)				
deviation of ve_{actual} from $ve_{expected}$	Reactive Plan		Non–Reactive Plan	
	t_f	v_f	t_f	v_f
0%	85.3518	100.0000	85.3518	100.0000
–0.5%	85.4227	100.0000	85.3518	99.7500
–1%	85.4942	100.0000	85.3518	99.4500
–2%	85.6393	100.0000	85.3518	99.0000
–5%	86.0911	100.0000	85.3518	97.5000
+0.5%	85.2816	100.0000	85.3518	100.2500
+1%	85.2120	100.0000	85.3518	100.5000
+2%	85.0747	100.0000	85.3518	101.0000
+5%	84.6773	100.0000	85.3518	102.5000

Table 5–2. Effects of variations in exhaust velocity on final velocity reached in reactive plan and non–reactive plan.

EFFECTS OF VARIATIONS IN BURN RATE (m_e) and EXHAUST VELOCITY (v_e)					
deviation of me_{actual} from $me_{expected}$	deviation of ve_{actual} from $ve_{expected}$	Reactive Plan		Non–Reactive Plan	
		t_f	v_f	t_f	v_f
0%	0%	85.3518	100.0000	85.3518	100.0000
–0.5%	–0.5%	85.5002	100.0000	85.3518	99.4794
–1%	–1%	85.6507	100.0000	85.3518	98.9618
–2%	–2%	85.9584	100.0000	85.3518	97.9355
–5%	–5%	86.9380	100.0000	85.3518	94.9272
+0.5%	+0.5%	85.2056	100.0000	85.3518	100.5235
+1%	+1%	85.0614	100.0000	85.3518	101.0500
+2%	+2%	84.7792	100.0000	85.3518	102.1119
+5%	+5%	83.9784	100.0000	85.3518	105.3695

Table 5–3. Effects of variations in burn rate and exhaust velocity on final velocity reached in reactive plan and non–reactive plan.

tive plan, which uses its sensitivity to the execution environment to avoid a priori computation. The only way for a classical planner to guarantee accurate a priori computation is to make no assumptions and instead identify and take into account all the relevant factors—a process likely to drive the system outside the realm of reasonable computation. The planner in the integrated approach, by reasoning about achievability and deferring certain planning decisions, is able to overcome the limitations of a pure inference-guided planner in real-world domains by taking advantage of execution-time information.

Chapter 6
RELATED WORK

This thesis has described an explanation–based learning approach towards integrating reactivity into a classical planner. This approach is one attempt towards solving the limitations of classical planning in complex, dynamic real–world domains. It also provides an explanation–based learning account of how general completable reactive plans may be learned from observation. This chapter is divided into three sections. The first section discusses other approaches involving the integration of planning and execution, including reactive planning and various reactive augmentations to classical planning. The second section presents related work on learning how to plan reactively. Finally, other work in the related topics of integrating perception into planning, deferring planning goals, the generalization–to–N problem in explanation–based learning, and using qualitative reasoning in planning is discussed in the third section.

6.1 Integrating Planning and Execution

The limitations of classical planning in dealing with complex, dynamic, real world domains have been convincingly argued many times in previous work [Agre87, Fikes72, Firby87, Schoppers87, Suchman87, Wilkins88] and we will not repeat those arguments here. Three broad approaches to these limitations have been developed. The first advocates the idea of pure reactive planning [Agre87, Brooks87, Firby87, Schoppers87, Suchman87], developed as an alternative to classical planning in dynamic real–world domains where a priori planning is either impossible or undesirable. The second involves the augmentation of classical planning with execution–time monitoring and failure recovery mechanisms [Fikes72, Wilkins88]. And the third investigates the division of planning responsibilities between an a priori planner and a reactive component [Cohen89, Turney89].

6.1.1 Reactive Planning

Much recent work has been within the alternative planning perspective of *reactive planning*, which characterizes the actions of an intelligent agent as simply being reactions to environmental stimuli [Suchman87]. In this perspective, a priori inference is replaced with reactions driven by execution–time information, enabling a system's planning to be sensitive to complex, dynamic execution environments. Agre and Chapman propose pure reactive planning in their work on *situated activity* [Agre87], as does Brooks in his work on *insects* [Brooks87]. In this planning paradigm, goals are represented only implicitly and reasoned about indirectly through the situation–action rules which govern an agent's behavior. In Firby's reactive planning approach [Firby87], some ability to explicitly represent and reason about goals is retained. Planning goals are hierarchically organized in task nets, and execution–time information is used to guide the planning process in determining which goals to attend to and consequently which actions to execute.

While a classical planner relies on a priori inferencing in its problem–solving, a reactive planner relies on its reactions to the environment to achieve its goals. The proofs of correctness constructed by a classical planner in its problem–solving provide guarantees that the plans constructed will achieve the goals. A reactive planner can provide no such guarantees. Without the ability to project the effects of its actions into the future, a reactive planner cannot reason about any possibly conflicting results of its actions. Its hill–climbing approach to planning may result in sub–optimal plans delaying goal achievement. Worse yet, a reactive planner may never achieve its goals if it gets caught in local maxima. In the integrated approach, the provably–correct nature of classical plans is combined with the execution–environment sensitivity of reactive plans. This approach extends classical planning by allowing certain planning goals to be deferred for execution. A goal may be deferred only if an achievability proof can be constructed, guaranteeing that the goal will be satisfiable during execution. Plans thus remain provably–correct. At the same time, execution–time information may influence the planning process since deferred goals are addressed during execution. Thus, a planner in the integrated approach is also sensitive to the execution environment.

Another major limitation of the reactive planning approaches discussed above is that they require the domain knowledge governing the reactions to be carefully constructed. Different problems requiring different behaviors necessitate different sets of rules determining the reactions. The set of rules used to determine reactions must be hand–tailored to achieve the desired problem–solving behavior. In his work on universal plan synthesis [Schoppers87], Schoppers provides an alternative reactive planning approach in which the rules governing the reactions are automatically constructed. Given a planning problem, represented by a conjunctive set of goals, a planner in this approach transforms its domain knowledge into a form enabling the direct determination of reactions. It constructs a *universal plan*, which contains advice regarding the appropriate reactions for every situation which may be encountered in the course of achieving the goals. A universal plan thus embodies a complete set of possible plans for a particular planning problem, accounting for all possible initial and intermediate states leading to a goal state.

Schoppers's approach allows for the automatic generation of reactions with their appropriate contexts from domain knowledge in a form such as that which might be used by a classical planner. It also provides a useful abstraction of the reactive planning space. However, in moving the entire planning process to execution time, the advantages gained by the power of inference in a classical planner are lost. While a classical planner is able to focus its attention on a single plan achieving some goal state from a particular initial state, Schoppers's planner must construct a universal plan which covers all possible initial states and all action sequences leading to goal states. If there exists a plan for achieving a particular goal, a universal plan will be able to achieve the goal, since it embodies a complete set of classical plans. However, while a classical planner is also able to detect when goal achievement is impossible, Schoppers's planner is not. Because it does not verify the satisfaction of any preconditions during planning, it may construct a universal plan and start execution, only to loop forever in its attempt to achieve an unachievable goal. In these cases, not only does a universal planner have to go through the effort of constructing an all–encompassing universal plan, but it does so needlessly. While some planning decisions might best be made during execution when further information is available, certain planning decisions can and should probably be made prior to execution to provide some guarantee of success. In the integrated approach, the contexts requiring or allowing certain types of goals to be deferred are learned by analyzing examples provided by an expert and determining the achievability conditions under which certain goals can be guaranteed to be satisfiable during execution. Thus, unlike reactive planners, a planner in the integrated approach learns to differentiate between goals which can and should be addressed prior to execution and goals which are either impossible or very difficult to plan for a priori but can easily and successfully be addressed during execution. The integrated approach is a compromise which retains the goal-directedness of classical planning and gains some of the reactive abilities of reactive planning.

6.1.2 Execution Monitoring and Failure Recovery

One of the ways in which a classical planner can be modified to enable it to deal with real-world domains is through the addition of execution monitoring and failure recovery mechanisms. One of the earliest reactive extensions to classical planning was the execution component PLANEX of the STRIPS system [Fikes72]. Using the generalized triangle tables generated by STRIPS, PLANEX employs a planning and execution strategy which involves repeatedly working backwards from the goal to the nearest unachieved subgoal. PLANEX is thus able to take advantage of fortuitously achieved subgoals as well repeat plan actions to reachieve subgoals which may have been clobbered by unforeseen circumstances. In the best case, PLANEX detects an already achieved goal, requiring no actions to be executed. In the worst case, PLANEX signals the total failure of a plan and requires complete replanning by STRIPS. Wilkins further addresses the problem of plan failure and subsequent error recovery in his SIPE system [Wilkins88]. SIPE monitors the execution of a plan for unexpected situations, and replans if necessary by using a general set of replanning actions to revise the portions of the executing plan which it deduces to be affected by the unforeseen circumstances. SIPE's replanning capabilities are geared towards retaining as much of the original plan as possible, which may often provide substantial savings over complete replanning but in certain cases may instead lead to non-optimal planning, both in the use of planning resources and in the achievement of the goal. While both PLANEX and SIPE have some ability to modify their plans during execution, they remain classical planners in that they begin execution with complete plans. To the extent that unforeseen circumstances occur, the final plan may not, in fact, achieve the goal.

The integrated approach provides a classical planner with the option of deferring goals for which it can construct achievability proofs guaranteeing their eventual achievement during execution, and thus may begin execution with a partial but completable plan. Execution is currently monitored only to detect the conditions enabling the determination of precise values for the conjectured variables corresponding to deferred goals. The detection of fortuitously achieved subgoals is limited the deferred, and hence monitored, goals. There are also no facilities for replanning, the primary focus being the problem of learning reactive plans. However, since the integrated approach retains much of the structure of a classical planner, the failure recovery methods developed for extended classical planners such as SIPE should also be applicable.

6.1.3 Combining A Priori Planning and Reactivity

A third approach to solving the limitations of classical planning in the real world involves the combination of a priori planning and execution-time reacting. In this approach, a classical planner is extended through the addition of a reactive component which is responsible for making certain types of planning decisions. Among the primary concerns in this approach are the division of planning responsibilities between the planning and reactive components and the development of control strategies by which a system's planning and execution are intertwined.

The SEPIA system of Turney and Segre [Turney89] alternates between *planning* and *improvisation*, with the planner imposing plan step orderings and the improvisor choosing the actions to be executed. The two components are able to affect each other through the constraints they enforce in the assumption-based truth-maintenance system (ATMS) SEPIA uses to maintain a current world model. With its ability to react to its execution environment through its improvisor, SEPIA is able to plan successfully in uncertain real-world domains wherein classical planning often fails due to its reliance on a complete and correct domain theory. Another method for integrating planning and reactivity can be found in Cohen et al's Phoenix system [Cohen89], which constructs fire-fighting plans for simulated forest fires. Plan construction and execution is achieved in Phoenix through a hierarchical organization of agents, with each agent consisting of a *cognitive component* and a *reflexive component*. The reflexive component reacts directly with sensory input to deal with unforeseen circumstances as well as fine-tunes the higher-level plans provided by the cognitive component. Reflexes enable a Phoenix agent to prevent certain disastrous events while fine-tuning removes some of the cognitive component's reliance on a perfect domain theory.

While SEPIA and Phoenix are able to extend classical planning into domains it would otherwise be unable to handle, their performance is greatly dependent upon the division of labor between their non-reactive and reactive components. The quality of SEPIA's solutions depends upon the strategy used to switch control between the planner and the improvisor as well as the predefined heuristic strategy the improvisor uses to determine SEPIA's actions. Phoenix's ability to manage forest fires depends upon the kinds of reflexes provided to its agents as well the coordination of and communication between the planning of its multiple agents. Much like the rules used to determine reactions in reactive planners, planning knowledge such as the strategies in SEPIA and the reflexes in Phoenix are greatly problem-dependent and must be provided to the system to achieve the desired performance, obstructing inter-task knowledge transfer. In the integrated approach, the criterion used to determine whether a planning decision is to be made reactively or non-reactively is that of achievability. A goal may thus be deferred only if that goal is guaranteed to be achievable during execution, where achievability proofs are constructed using rules derived from domain-independent rule schemata. Although the implemented system currently cannot derive the rules from schemata on its own, rule schemata do provide an implementor with problem-independent principles by which the division of planning responsibilities can be made between the non-reactive and reactive components of a planning system.

Another reason this dependence on predefined control and heuristic strategies does not arise in the integrated approach is because this approach investigates the integration of planning and execution only within a single plan corresponding to a single set of goals. SEPIA and Phoenix deal with a more complex level of integration, allowing for multiple goals and multiple plans as well as partial plan construction and execution. These approaches focus on the division of certain planning responsibilities between the non-reactive and reactive components, while the integrated approach focuses on the division of the whole planning process between the a priori partial plan construction phase and the execution-time plan completion phase. The predefined division of labor between a Phoenix agent's cognitive and reflexive component is precisely the kind of division learned in the integrated approach.

By alternating between planning and execution, SEPIA and Phoenix also provide a solution to the problem of uncertainty due to sensor or effector errors. Flawed beliefs can be corrected, imperfect action executions remedied, and an up-to-date model of a dynamic world maintained. The integrated approach currently addresses the problem of uncertainty only indirectly through the notion of achievability, which enables planning for certain goals for which a planner relying on perfect knowledge would otherwise require a domain theory likely to be intractable. The current focus is only on determining which goals should be planned for prior to execution and which should be planned for during execution. Thus planning decisions are irreversible and assumed to be correct, and there is no need for constant plan modification.

To summarize, SEPIA and Phoenix differ from the integrated approach in three ways. First, the division of planning responsibilities between non-reactive and reactive components must be user-specified in SEPIA and Phoenix but is learned in the integrated approach. Second, the integrated approach investigates the integration of planning and execution within a single plan while SEPIA and Phoenix investigate integration between multiple plans. Third, the problem of uncertainty is attacked directly through constant plan modification in SEPIA and Phoenix but only partially and indirectly through deferred goals in the integrated approach.

6.2 Learning to Plan Reactively

Most of the work in learning to plan reactively has been within the reactive planning framework viewing acting as actually reacting to environmental stimuli in a manner which will eventually result in the achievement of goals. Thus, work in learning reactive plans has been towards learning appropriate reactions to different situations. The Theo-Agent system of Blythe and Mitchell [Blythe89] presents an explanation-based learning method by which situation-action rules are learned for use in planning in a robot domain. Theo-Agent can decide on its actions through inferencing as well as through the use of stimulus-response rules which directly determine actions to execute based upon the current goals and the current state. Plans constructed through inference are used by Theo-Agent in operationalizing certain target rules into stimulus-response rules which the system can then use directly in later executing the same plan. Thus, the ultimate fate of Theo-Agent seems to lie in the transformation of its entire domain knowledge into a complete set of stimulus-response rules. Unlike the universal plans in Schoppers's approach [Schoppers87], however, the set of reactions of Theo-Agent does not have to be complete, since it is the system's planning experience which warrants the formation of the stimulus-response rules. Theo-Agent's approach to plan execution is similar to that of PLANEX [Fikes72], with the main difference being that in deciding on its actions Theo-Agent moves forward from the initial state while PLANEX goes backward from the goal. At execution time, both verify the conditions supporting the planning decisions made a priori, with the result being that fortuitous situations can be taken advantage of, action steps can be repeated if the goals they achieve are somehow clobbered during execution, and replanning can be triggered if the plan being executed by PLANEX or the stimulus-response rules corresponding to a plan in Theo-Agent cannot achieve the goal. A current problem with Theo-Agent lies in its optimistic planning strategy, which assumes unverifiable effects of previous actions to be true in planning for subsequent actions. If such assumptions prove faulty, the stimulus-response rules derived from a plan depending on them will be incorrect, causing Theo-Agent to make wrong planning decisions and possibly inefficient and even unsuccessful planning. This is the same hill-climbing problem faced by pure reactive planners which do not look into the future and may thus make costly or fatal mistakes. Current work on Theo-Agent is addressing this problem.

The main difference between the integrated approach and Theo-Agent is that the integrated approach insists on guaranteed plans while Theo-Agent plans optimistically in the absence of verifying information. Thus, while Theo-Agent may be prone to the hill-climbing behavior of pure reactive planners due to its inability to differentiate between satisfiable and unsatisfiable plan preconditions, a planner in the integrated approach guarantees the achievement of its deferred goals through achievability proofs. On the other hand, while Theo-Agent is able to plan even in the face of incomplete information, a planner in the integrated approach will be unable to construct plans for which it cannot prove the achievability of deferred goals. The ideal system probably lies between these two extremes, in a planning strategy requiring plausible but not necessarily absolute proofs of achievability of unverifiable preconditions or deferred goals.

6.3 Related Topics

The primary goal of this research is to develop an integrated approach to planning and learning reactive plans. However, this work also borrows from the work in several other research areas. One of the motivating factors for this research is that accurate inferencing about the real world is a very difficult task and more accurate modeling of the real world is often made possible by using information gathered through perception during execution. Another key issue is the deferment of goals which cannot be planned for prior to execution, before the gathering of certain information. A third related issue is the generalization-to-N problem in explanation-based learning, for which the integrated approach provides a reactive solution to those problems involving repeated actions. Lastly, there is the idea of using qualitative reasoning in planning to provide a larger grain-size of reasoning.

6.3.1 Integrating Perception into Planning

One way in which perception has been used to solve the inferencing problem is as a verification mechanism for plan inferences [Brooks82, Doyle86]. Doyle et al's GRIPE system analyzes a plan to determine where verification is needed, and generates anticipated sensor values together with perception requests which it then inserts into the plan. Similarly, Brooks uses error propagation based on uncertainty modeling to determine where monitoring is necessary in a plan. In both these approaches, perception is used solely as a tool for error detection and recovery. Modeling the real world is accepted as uncertain at best, thus requiring execution-time verification for the conditions supporting a priori planning decisions. In contrast, in the integrated approach execution-time sensory input may be used in the construction of a plan itself. The execution environment is not monitored to verify that a particular action is executable or has been executed successfully, but rather to determine appropriate completions to the partial plans constructed prior to execution.

6.3.2 Deferring Goals

The idea of deferring planning goals is also a central issue in hierarchical planning [Iwasaki85, Sacerdoti74, Sacerdoti77, Stefik81]. In this perspective, planning is characterized as taking place through successively less abstract levels of plan representation, where *abstraction* may refer to the planning process, plan representation, or a combination of the two. Goals in higher level of abstractions may thus be deferred for lower levels. One way in which the integrated approach may be viewed is as an extension of hierarchical planning, with the lowest level corresponding to an execution–time level of plan representation. However, the integrated approach requires the goals deferred into this lowest level to first be proven achievable—a constraint not present in hierarchical planning. This is, however, due primarily to orthogonal research objectives. Hierarchical planning provides a planning framework in which goals can be prioritized for the construction of complete plans prior to execution, whereas the aim of this work is to develop a framework for learning in which goals can be deferred in constructing completable reactive plans.

Opportunistic planning [Hayes–Roth79] is another planning perspective in which planning goals may be deferred. An opportunistic planner suspends goals until opportunities for planning for them arise. Hammond [Hammond89] extends the notion of opportunism into execution by allowing opportunities to dictate the introduction of new goals, activation of suspended goals, and suspension of current goals during execution. In opportunistic planning, goals are deferred or suspended when the system becomes unable to plan for them further with the current state of its knowledge. In contrast, goals in the integrated approach are deferred for execution when they are best planned for because of the additional information which becomes available then. Furthermore, the deferred goals in the integrated approach are guaranteed to be achievable whereas the suspended goals in opportunistic planning may never be reactivated and achieved. The decision to suspend and reactive goals is made in a bottom–up fashion in opportunistic planning, with the execution environment determining which goals are to be attended to. In the integrated approach, goals are deferred and reconsidered in a top–down manner, with the planner dictating which goals are to be deferred as well as how to attend to them during execution. Another difference between the opportunistic planning approach and the integrated approach is in the use of execution–time information. In opportunistic planning, execution–time information directly controls the planning process by determining the opportune times to plan for different goals, any of which may be suspended or deferred if further planning becomes impossible. In the integrated approach, execution–time information is used directly in determining appropriate plan completions to plans which are guaranteed to be completable by the achievability proofs associated with its deferred goals.

6.3.3 Solving the Generalization–to–N Problem

One type of completor discussed in the chapter on learning general completable reactive plans (Chapter 5) is the iterator, which executes an action or action sequence repeatedly until a particular goal is achieved. This approach for executing repeated actions can also be viewed as one solution to the generalization–to–N problem in explanation–based learning. Various solutions to the issue of generalizing explanation structure been developed in the form of rule instantiation sequences [Shavlik87] and proof automata [Cohen87]. Both of these enable the use of a single generalized structure to construct different plans involving different numbers of repetitions of particular actions. However, planners in these approaches still fall within the classical planning framework in which plans are completely constructed prior to execution. The integrated approach proposes an alternative solution wherein the number of repetitions of a particular action or action sequence may be computed dynamically by repeatedly assessing the execution environment and deciding whether an action should be repeated by determining whether or not the goal has been reached. The difference can thus be characterized as the time at which plan *unfolding* takes place—prior to execution in the Shavlik and Cohen approaches and during execution in the integrated approach. Neither approach is necessarily superior in that each offers certain advantages more useful for particular types of planning tasks. For example, while deciding how many times to pound a nail to drive it through wooden planks is probably a planning decision best made during execution, setting a table is probably best done by determining a priori the number of places at the table and consequently the number of times the action sequence of laying out a place setting should be performed.

6.3.4 Using Qualitative Reasoning in Planning

The idea of using qualitative reasoning in planning has been investigated in other work [DeJong89, Forbus89, Hogge87]. Hogge and Forbus present ways in which qualitative reasoning work can be extended for use in planning domains. Hogge's operator compiler transforms Qualitative Process Theory [Forbus84] representations of processes into rules and operators for use by a temporal interval–based planner. Forbus presents an alternative approach using action–augmented envisionments, which allows a qualitative reasoning system to reason about actions and their possible effects by varying the set of manipulable assumptions envisionments depend upon. Their work is complementary to this work in learning reactive plans in that their aim is to develop ways in which qualitative reasoning can be used in planning, and qualitative reasoning was used to provide a more intuitive level of reasoning for proving achievability in the integrated approach to planning. DeJong also uses qualitative reasoning as a knowledge representation tool in his work on learning how to plan in continuous domains—i.e. domains involving multiple, overlapping, time–spanning processes which make the use of situation calculus unwieldy. As with the integrated approach, qualitative reasoning is used in planning to provide a higher level of reasoning which allows for the expression of time–spanning monotonic changes of quantity values, and qualitative proportionalities and influences between quantities.

Chapter 7
CONCLUSIONS AND FUTURE WORK

This thesis has presented an explanation–based learning strategy for learning reactive plans for use in an integrated approach to planning. In this approach, a classical planner is augmented with the ability to defer achievable goals and to address them during execution. This is achieved through the use of conjectured variables and achievability proofs, which enable the construction of valid contingent explanations which can then be generalized into general completable reactive plans. In the following sections, the major ideas presented by this work are summarized, together with a discussion of the limitations of the current approach and directions for future work.

7.1 Achievability Proofs

The integrated approach to planning revolves around the notion of achievability. Proofs of achievability enable goals to be deferred until execution without sacrificing the provably–correct nature of plans. Furthermore, proving achievability independently of determining the actions to achieve the associated goal simplify the planning process by allowing planning for certain goals which are very difficult to plan for a priori to be deferred until execution when additional information may be gathered and used in making planning decisions. Three classes of problems for which achievability proofs can be constructed independently of determining actions were presented in this work. These were problems involving: 1) repeated actions and terminal goal values, 2) continuously–changing quantities and intermediate goal values, and 3) multiple opportunities. While these three problem classes span a wide range of planning problems, they do not encompass all the achievable goals of interest. Current work is addressing the problem of identifying other problem classes, although it is expected that a small number of classes will be sufficient to characterize the problem space attacked by the integrated approach to planning.

The major limitation of achievability as it has been defined in this work is that it is required to be absolute—i.e. in proving the achievability of a goal, the existence of a plan achieving a goal must be proven absolutely. More realistically, however, achievability proofs might be probabilistic. Take, for example, the problem of crossing the street. While you may very well believe that you can detect an appropriate gap between cars through which you can cross safely and reach the other side of the street, there is always that chance that the street has been cordoned off for the digging being done for pipe–laying, or that a lunatic driver will try to run you over, or that you will be beamed off the planet onto an alien spaceship. Proving that a particular goal is absolutely achievable means providing a one–hundred percent guarantee that a plan achieving it will be determined and executed successfully to achieve the goal. While the integrated approach permits a different kind of characterization of domains minimizing a priori computation and eliminating the need for certain kinds of information prior to execution, the task of constructing an absolute achievability proof will still require a perfect domain theory and complete reasoning. The problem of intractability again surfaces.

A probabilistic characterization of achievability would provide a more realistic planning perspective, allowing a planner to reason about degrees of achievability and to make planning decisions based on them. A planner could, for example, always opt for the plan with the deferred goals which are most likely to be achievable. Or it could choose a plan with less achievable goals if a plan with more achievable goals would cost much more with relatively little gain in achievability. There is also the issue of determining which factors are worth taking into consideration in proving achievability. For example, in determining the availability of a paper cup for coffee in a particular cabinet, one would probably want to account for the number of people who may have used coffee cups during the day, but not for the possibility that a bottle of acid in the lab directly above the lounge crashed to the floor, allowing the acid to make a hole through the ceiling and into the cabinet, and eat its way through one side of the coffee cups. The problem of discriminating between useful and erroneous assumptions is the problem primarily addressed by research on incremental learning algorithms based on assumptions [Bennett89, Chien89, Ellman88, Mostow89], and thus the work there may provide direction in this matter.

Reasoning about achievability is important for another reason—sometimes, a planner may take steps to increase goal achievability. For example, on a long drive from Chicago to New York, one may decide to defer the planning decision of precisely when and where to stop for the night since there are many possibilities for achieving the goal of finding a place to sleep. However, it may be that in the city the traveller eventually chooses to stop at, that night turns out to be fully-booked for all the hotels, rendering the goal unachievable and the plan a failure. If, instead of deferring the hotel decision, the traveler calls ahead of time and makes a reservation, then he greatly increases the chance of his finding a place to sleep on his road trip. The goal is still not absolutely achievable, since other unexpected circumstances may still come into play—the reservation may get lost, a fire may destroy the hotel, or the traveler may encounter car trouble and not even make it past the Illinois border—but the goal has certainly become more achievable. The increase in achievability does not, however, come without its own costs. The traveler must now make the a priori decisions of precisely which city to stop at and which hotel to stay in and take these decisions into account during a priori planning. The resulting plan will also be less flexible. The investigation of various tradeoffs in planning costs and degree of achievability is another interesting area for future work.

7.2 Reactive Plans

The completors incorporated into general plans serve to find a completion for a plan during execution by determining appropriate values for conjectured variables. Three types of completors corresponding to the three classes of achievable goals were introduced in this work: 1) iterators for repeated actions and terminal goal values, 2) monitors for continuously-changing quantities and intermediate goal values, and 3) filters for multiple opportunities. All three are fairly simple, involving minimal computation and making few planning decisions.

Consider, however, the problem of hammering a nail partway into a piece of wood. At first glance, this may not seem too different from hammering a nail all the way into a piece of wood which has been shown to have a corresponding achievability proof. However, this achievability proof relies on the goal value being a terminal, non-surpassable goal value, which is not the case in the hammering partway problem. The intermediate value achievability proof is of no help either, since this relies on a continuously-changing quantity, whereas hammering involves a series of discrete, monotonic changes in the depth the nail is embedded in the wood.

The simplicity of the constraints imposed by the intermediate value achievability proof gives rise to related problems. This proof only guarantees that a particular intermediate value will be achievable—it has nothing to say about how long a plan will take to achieve it. Thus, a particular higher goal velocity will be deemed possible whether velocity increases at the rate of a mile a minute or a yard a year. Perhaps a more serious problem is that purely qualitative achievability proofs such as those currently constructed by the system make planning for interacting goals a very difficult task. For example, docking a spaceship will involve not only monitoring the decreasing velocity but also the decreasing distance between the spaceship and the landing dock. These two quantities are not independent and an achievability proof for the docked state must take this interdependence into account, a requirement very difficult to deal with purely qualitatively.

The solution to problems such as hammering a nail partway into a piece of wood, achieving reasonable acceleration, and docking a spaceship successfully may be in providing the system with some quantitative reasoning knowledge. The objective is not to enable complete quantitative reasoning using perfect but intractable mathematical models, but rather to find an integration of quantitative and qualitative reasoning knowledge which would provide a more powerful reasoning system without getting bogged down by too much computation.

A consequence of this would be the need for more complex completors with more runtime responsibility. A plan for hammering a nail partway into a wooden plank, for example, might involve a completor which signals when the goal state is nearing so that the force of the hammer blow could be decreased to minimize the chance of going past the goal state. A plan for docking a spaceship might involve a completor which monitors both the spaceship velocity and distance to the landing dock, compares this to the expected values, and makes adjustments as necessary. The addition of quantitative reasoning would greatly broaden the scope of the integrated approach, and with it come a variety of new problems for future work.

7.3 Contingent Explanation–Based Learning

The contingent explanation–based learning algorithm presented in this thesis enables the learning of general reactive plans from observation. Deferred goals are represented through the use of conjectured variables, which serve as placeholders for the eventual values of plan parameters, the determination of whose values are deferred until execution. Accompanying achievability proofs guarantee that values will be found for conjectured variables. Each of the three types of completors presented in this work is associated with a particular conjectured variable. An iterator uses a conjectured variable for the number of repetitions of an action. A monitor uses a conjectured variable for the end of the time interval over which certain conditions hold. A filter uses a conjectured variable for the object which will eventually be chosen from a group of objects.

More complex problems will probably require plans involving more complicated types of deferred goals. For example, imagine a pest exterminator planning to de–bug a house. Proving the achievability of a pest–free state may involve proving that he will be able to get rid of all the different kinds of bugs in the house. Proving that does not mean having to know exactly which types of creatures are in the house. Instead, the exterminator may simply know that he has every kind of gadget and poison needed to get rid of every house pest known to man. Thus, he may defer the choice of extermination method—the precise sequence of actions he will use to do the job—until he is at the house and has identified the pests. This deferred goal involves much more than the deferment of finding a value for a parameter of an operator—it involves deferring the choice of the operator, the objects with which the operator is to be performed, and the state in which the operator will be executed. To address these kinds of deferred goals and the plans that generate them, a more general notion of conjectured variables is needed.

7.4 The Planning Problem

In the chapter on the planning problem, it was mentioned that the primary difference between the a priori phases of classical planning and the integrated approach was the option of proving achievability instead of determining a plan in the integrated approach. This suggests an alternative characterization of the planning problem, no longer to be taken to imply determining a precise sequence of actions but instead only the existence of some sequence of actions—i.e. planning ≡ proving achievability. This has various ramifications, the most important of which is that proving achievability becomes no longer an option in the planning process but instead plan determination becomes an option in proving achievability. An advantage of such a characterization would be a more uniform treatment of planning. All explanations would start out as contingent explanations, with all conjectured variables eventually either supported by achievability explanations or replaced by precise values and accompanying subplans, blurring the distinction between classical (non–reactive) and reactive plans. It might thus be possible to learn a general reactive plan from either an example of a non–reactive plan or a reactive plan, and learned non–reactive plans may be modified into reactive plans through experience.

Viewing the planning problem as the problem of proving achievability has another interesting effect—planning can be characterized as a tradeoff problem of maximizing the chance of success while minimizing cost. Currently, the only concern is maximizing (in fact, guaranteeing) success. Planning resources are unlimited and thus planning is assumed to take place instantaneously, both prior to execution and during execution. For example, in the spaceship acceleration example, in the initial state the spaceship is travelling at the velocity of 65 m/s and the plan to achieve a velocity of 100 m/s includes a monitor which invokes the stop–fire–rockets action when the goal velocity is reached. In the time taken to construct this plan, the state of the world is assumed to remain the same, and the time it takes for the monitor to detect the goal velocity and execute the stop–fire–rockets action is assumed to insignificant. But in fact planning and execution take time and the real world changes with time. The spaceship planner may be constrained to find a plan to achieve the acceleration quickly to avoid a meteor about to cross its path, and it may be crucial to achieve the goal velocity precisely to remain within a certain range behind another spaceship. Work on real–time planning [Dean88, Shekhar89] may provide direction in addressing the issues this brings up such as implementing real–time constraints on planning and execution, reasoning about planning time and planning costs, reasoning about execution time and execution costs, and optimizing plans based on certain criteria.

7.5 Psychological Relevance

One of the primary motivating factors for this work came from observing the way people plan. People are not classical planners, perhaps because the real world just cannot be perfectly predicted. Neither do they act purely reactively for problems in a wide variety of well–behaved domains. Oftentimes some a priori planning is possible and indeed desirable—in building a house, for example, or vacationing in a foreign country, or financing a corporate buy–out. People rarely determine everything a priori, but they do make many decisions prior to execution. Furthermore, they are very good at prioritizing planning decisions and weeding out decisions which can be made at execution time without sacrificing the integrity of a plan.

One of the areas for future work would be in psychological experiments investigating the integrated approach to planning presented here. Such experiments would be useful not only in validating this approach, but also in providing insight into the kinds of planning decisions people tend to defer and the reasons behind such deferment. It has been interesting to note from informal discussions with various people (particularly some very optimistic officemates) the various kinds of *achievability explanations* one can come up with for different problems. These have included reasons such as: "Because I've done it before," "Because I know if something goes wrong I can [remedy the situation," and "Because I can live with [the worst case]"—reasons which involve a different kind of reasoning about achievability than has been discussed in this thesis. While most may never be directly helpful in proving achievability for the kinds of problems a computer implementation would be useful for, they are invaluable not only in motivating this work but also as inspiration for ideas about achievability.

7.6 Conclusion

The main advantages of a classical planner are its construction of plans with a high likelihood of success, its goal–directedness, and its ability to learn from experience. The main advantages of a reactive planner are its sensitivity to the execution environment and its non–reliance on perfect a priori information. The integrated approach to planning presented in this thesis allows for the a priori construction of provably–correct partial plans, which can be completed dynamically using execution–time information. Contingent explanation–based learning enables the learning of general completable reactive plans, which allows for the cost of constructing a reactive plan to be amortized over the number of times that plan is used. Thus, while the integrated approach incurs the additional cost of a priori planning over reactive planning, in the long run this cost is well worth the goal–directedness and guarantees of success afforded by a priori planning. The integrated approach also retains the benefits of classical planning without inheriting its intractability disadvantage. A planner in the integrated approach constructs provably–correct plans but reduces its need for perfect a priori information by deferring goals and gathering additional information during execution for use in making planning decisions. The integrated approach to planning and the contingent explanation–based learning strategy presented in this thesis thus provide a powerful new method by which a planner can learn to deal effectively with a wide variety of real–world problems.

Appendix A
DERIVING THE INTERMEDIATE VALUE RULE

Achievability proofs are implemented in the form of second–order predicate calculus rules called rule schemata from which first–order rules can be derived for use in constructing achievability proofs. Examples of rule schemata are shown in the chapter on achievability (Chapter 2) and an example of a rule derived from a rule schema is used in the example presented in the implementation chapter (Chapter 5). Here, the derivation of that intermediate–value rule from the Intermediate Value Schema is presented as an example of how rules can be derived from rule schemata.

The Intermediate Value Rule Schema states that if q is a continuous quantity having a value v0 at some time t0, and certain conditions θ being true over some interval [t0 t2] will result in q having some greater value v2 at time t2, then for all values v1 between v0 and v2 there exists some time t1 within the interval [t0 t2] such that if θ holds over the interval [t0 t1], q will have the value v1 at t1:

Intermediate Value Rule Schema

```
[∀ θ
    [∀ q v0 v2 t0 t2
        (   (value q v0 t0) AND
            (continuous q) AND
            ((θ (t0 t2)) → (value q v2 t2)) )
    →
        [∀ v1
            ((between v1 v0 v2)
            → [∃ t1 ((within t1 (t0 t2)) AND ((θ (t0 t1)) → (value q v1 t1)))] ) ] ] ].
```

In the problem of spaceship acceleration, we are interested in reasoning about intermediate values of monotonically–changing quantities. The variable θ can be replaced by a predication about the qualitative behavior of q over the time interval (t0 t2) to yield the following intermediate value rule:

Intermediate Value Rule .

```
[∀ q v0 v2 t0 t2 beh
    (   (value q v0 t0) AND
        (continuous q) AND
        ((qualitative_behavior q beh (t0 t2)) → (value q v2 t2)) )
→
    [∀ v1
        (between v1 v0 v2)
        → [∃ t1 ((within t1 (t0 t2)) AND
                    ((qualitative_behavior q beh (t0 t1)) → (value q v1 t1)) ) ] ) ] ].
```

This rule states that if a continuous quantity q has the value v0 at time t0, and has a certain behavior beh over some time interval (t0 t2) which leads to q having the value v2 at time t2, then for every value v1 between v0 and v2, there is some time t1 within the interval (t0 t2) such that if q has the behavior beh over the interval (t0 t1), q will have the value v1 at time t1.

Prefacing the universally–quantified variables with "?" and the conjectured variables with "!" in the rule above, we have:

```
        (   (value ?q ?v0 ?t0) AND
            (continuous ?q) AND
            ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) )
    →
        (   (between ?v1 ?v0 ?v2)
            → ((within !t1 (?t0 ?t2)) AND
                ((qualitative_behavior ?q ?beh (?t0 !t1)) → (value ?q ?v1 !t1)) ) )
```

which can be rewritten into:

```
        (   (value ?q ?v0 ?t0) AND
            (continuous ?q) AND
            ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) AND
            (between ?v1 ?v0 ?v2) )
    →
        (   (within !t1 (?t0 ?t2)) AND
             ((qualitative_behavior ?q ?beh (?t0 !t1)) → (value ?q ?v1 !t1)) ).
```

This can be split into the following two rules:

Rule 1:

```
        (   (value ?q ?v0 ?t0) AND
            (continuous ?q) AND
            ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) AND
            (between ?v1 ?v0 ?v2) )
    →
        (   (within !t1 (?t0 ?t2) )
```

Rule 2:

```
        (   (value ?q ?v0 ?t0) AND
            (continuous ?q) AND
            ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) AND
            (between ?v1 ?v0 ?v2) )
    →
        (   ((qualitative_behavior ?q ?beh (?t0 !t1)) → (value ?q ?v1 !t1)) ).
```

Finally, Rule 2 can be transformed to yield the intermediate value rule of the acceleration example:

```
        (   (value ?q ?v0 ?t0) AND
            (continuous ?q) AND
            ((qualitative_behavior ?q ?beh (?t0 ?t2)) → (value ?q ?v2 ?t2)) AND
            (between ?v1 ?v0 ?v2) AND
            (qualitative_behavior ?q ?beh (?t0 !t1) )
    →
        (   (value ?q ?v1 !t1))  ).
```

Appendix B
ACCELERATION EQUATION USED IN THE NON–REACTIVE PLAN

In the implementation chapter of this thesis, an example involving the firing of spaceship rockets to achieve a higher velocity was discussed. In the classical planning mode of the implemented system, the length of the interval over which the rockets must be fired to achieve the desired velocity, and hence the time at which to stop the rockets, is determined prior to execution. This is done through the use of an equation derived from the principle of conservation of linear momentum. The following solution is adapted from the solution of a similar problem in ([Halliday81], p. 144).

Problem

Find a general equation for determining how long to fire the rockets of a spaceship in order to achieve a particular higher velocity from a particular initial velocity. Assume that the fuel burns at a constant rate and that the exhaust leaves the spaceship at a constant velocity.

Solution

Assuming that the total external force on the spaceship is zero, then the Principle of Conservation of Linear Momentum holds. This principle states that: *When the resultant external force acting on a system is zero, the total linear momentum of the system remains constant* ([Halliday81], p. 142). Let:

P	=	total system momentum
M	=	total mass of the spaceship
v	=	velocity of the spaceship
u	=	absolute exhaust velocity

Over some time Δt, the spaceship will lose some mass ΔM of fuel which will leave the spaceship as exhaust gases traveling at a particular velocity u. At the same time, the spaceship's velocity will increase by some Δv. Thus:

P_i	=	initial system momentum	=	Mv
P_f	=	final system momentum	=	$(M - \Delta M)(v + \Delta v) + \Delta M u$

So in this situation:

$$0 = \frac{\Delta P}{\Delta t} = \frac{[(M - \Delta M)(v + \Delta v) + \Delta M u] - [Mv]}{\Delta t}$$

This can be rewritten as:

$$M\frac{\Delta v}{\Delta t} = (u - v - \Delta v)\,\frac{-\Delta M}{\Delta t}$$

or as Δt approaches 0:

$$M\frac{dv}{dt} = (u - v - \Delta v)\,\frac{-dM}{dt} \qquad \text{(Equation 1)}$$

Let:

v_e	=	relative exhaust velocity=	$u - v - \Delta v$
m_e	=	burn rate of fuel	

M_i	=	initial mass of spaceship
v_i	=	initial spaceship velocity
v_f	=	goal spaceship velocity

Using the following equation for the mass of the spaceship as a function of time:

$$M = M_i - m_e t$$

Equation 1 can be rewritten as:

$$\frac{dv}{dt} = \frac{v_e m_e}{M_i - m_e t}$$

$$dv = v_e m_e \int \frac{1}{M_i - m_e t}\, dt$$

$$v_f - v_i = v_e \ln(M_i + m_e t) + C \qquad \text{(Equation 2)}$$

Solving this equation for the constant C at $t = 0$:

$$0 = v_e \ln M_i + C$$

$$C = -v_e \ln M_i$$

Solving Equation 2 for t, using C as above:

$$t = \frac{e^{\frac{v_f - v_i - C}{v_e}} - M_i}{m_e}$$

$$t = \frac{e^{\frac{v_f - v_i}{v_e} + \ln M_i} - M_i}{m_e}$$

$$t = \frac{e^{\frac{v_f - v_i}{v_e}} M_i - M_i}{m_e} \qquad \text{(Equation 3)}$$

This t is the length of time the rockets must be fired to achieve the goal velocity v_f from the initial velocity v_i. Equation 3 can be used in constructing the following classical or non–reactive spaceship acceleration plan:

Non–Reactive Plan

[**fire–rockets** at time t_i

wait for time t

where $t = \dfrac{e^{\frac{v_f - v_i}{v_e}} M_i - M_i}{m_e}$

given v_i = velocity at time t_i

v_f = goal velocity

v_e = exhaust velocity

m_e = burn rate

M = total mass of spaceship at time t_i

stop–fire–rockets at time $t_f = t_i + t$].

Appendix C
SAMPLE RUN

Following is a sample run of the system learning general acceleration plans. It is first run in the integrated mode, where it learns a general completable reactive plan from an example involving the acceleration of a spaceship. The system is then run in the classical planning mode, where it learns a general non-reactive plan from the same example. Italicized comments were inserted into the following dribble file after the actual run for explanatory purposes.

```
;;; Dribble file #P"thesis.drb" started
T

;;; INTEGRATED APPROACH MODE

> (clear-all)
Cache cleared
Working memory cleared
Nodes forgotten
Interval constraints removed
NIL
> (load-reactive)
;;; Loading source file "ar-r.lisp"

Rules loaded
;;; Loading source file "ae-r.lisp"

Example loaded
NIL
> *goal*
(MOVING SPACESHIP 100.0 (&TS &TG))
;;; The goal is to have the spaceship constantly moving at 100 m/s.

> (look-init-data)
Initial Data
     (SPACE_VEHICLE SPACESHIP)
     (LOCATION SPACESHIP DEEP-SPACE)
     (OBSERVABLE (VELOCITY SPACESHIP))
     (MAX_VAL (VELOCITY SPACESHIP) 500.0)
     (CONTINUOUS (VELOCITY SPACESHIP))
     (VALUE (VELOCITY SPACESHIP) 65.0 10.0)
     (OFF_ROCKETS SPACESHIP (-100.0 &T0))
     (AFTER &TS 10.0)
     (AFTER &TG 10.0)
NIL
;;; The spaceship is initially moving at 65 m/s at time 10.
> (run *goal* :wm t :full t)


Initializing working memory...

Attempting to learn plan(s) for (MOVING SPACESHIP 100.0 (&TS &TG))

;;; As the system observes every action, it attempts to construct an explanation for the action—i.e
;;;     verify that the action was executable.

Explaining actions...
```

```
   Observed (FIRE-ROCKETS SPACESHIP 10.0)
   Finding explanations...
   Found 1 explanation(s)

[5] (FIRE-ROCKETS SPACESHIP 10.0)
  [4] (FIRE-ROCKETS SPACESHIP 10.0)
     [1] (SPACE_VEHICLE SPACESHIP)
     [2] (OFF_ROCKETS SPACESHIP (-100.0 10.0))
     [3] (BEFORE -100.0 10.0)

   Observed (STOP-FIRE-ROCKETS SPACESHIP 17.1576)
   Finding explanations...
   Found 1 explanation(s)

[11] (STOP-FIRE-ROCKETS SPACESHIP 17.1576)
  [10] (STOP-FIRE-ROCKETS SPACESHIP 17.1576)
     [7] (SPACE_VEHICLE SPACESHIP)
     [8] (ON_ROCKETS SPACESHIP (10.0 17.1576))
      [6] (ON_ROCKETS SPACESHIP (10.0 17.1576))
         [4] (FIRE-ROCKETS SPACESHIP 10.0)
            [1] (SPACE_VEHICLE SPACESHIP)
            [2] (OFF_ROCKETS SPACESHIP (-100.0 10.0))
            [3] (BEFORE -100.0 10.0)
     [9] (BEFORE 10.0 17.1576)

All actions explained.  Attempting to explain goal (MOVING SPACESHIP 100.0 (&TS &TG))

Constructed 1 explanation(s) (*explanations*)

Extracting plan(s)...
   Determining plan components from [EXPLANATION for (MOVING ?SS9056 ?V19035 (?ST19062
?ET19064))]
Extracted 1 plan(s)    <Hit <CR> to continue...>
```

;;; The general plan is checked for conjectured variables, and an appropriate completing component
;;;   is incorporated into the general plan for every conjectured variable.

```
Looking for non-operational components
  Checking (!T9036) in (BEFORE ?T19017 !T9036)
  Checking (!T9036) in (ON_AFTER ?ST19062 !T9036)
  Checking (!T9036) in (STOP-FIRE-ROCKETS ?SS9056 !T9036)
Operationalizing non-operational components...
  Operationalizing !T9036
  Creating monitor for !T9036 using ((VALUE (VELOCITY ?SS9056) ?V19035 !T9036))
Integrating 1 monitor(s) into <plan 1>

Constructed 1 plan(s) (*plans*)
NIL
> (show-explanation (first *explanations*))


Specific explanation for (MOVING SPACESHIP 100.0 (&TS &TG)):
```
[44] (MOVING SPACESHIP 100.0 (&TS &TG))
```
   [26] (VALUE (VELOCITY SPACESHIP) 100.0 17.1576)
    [25] (VALUE (VELOCITY SPACESHIP) 100.0 17.1576)
       [13] (VALUE (VELOCITY SPACESHIP) 65.0 10.0)
       [14] (CONTINUOUS (VELOCITY SPACESHIP))
       [17] (-> ((QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) INCREASING (10.0
T+INF))) ((VALUE (VELOCITY SPACESHIP) 500.0 T+INF)))
        [16] (-> ((QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) INCREASING (10.0
T+INF))) ((VALUE (VELOCITY SPACESHIP) 500.0 T+INF)))
           [15] (MAX_VAL (VELOCITY SPACESHIP) 500.0)
       [18] (BETWEEN 100.0 65.0 500.0)
       [24] (QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) INCREASING (10.0 17.1576))
        [23] (QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) INCREASING (10.0
17.1576))
           [19] (ON_ROCKETS SPACESHIP (10.0 17.1576))
            [6] (ON_ROCKETS SPACESHIP (10.0 17.1576))
               [4] (FIRE-ROCKETS SPACESHIP 10.0)
                  [1] (SPACE_VEHICLE SPACESHIP)
                  [2] (OFF_ROCKETS SPACESHIP (-100.0 10.0))
                  [3] (BEFORE -100.0 10.0)
           [22] (NEGLIGIBLE (EXTERNAL_FORCE SPACESHIP) (10.0 17.1576))
            [21] (NEGLIGIBLE (EXTERNAL_FORCE SPACESHIP) (10.0 17.1576))
               [20] (LOCATION SPACESHIP DEEP-SPACE)
   [38] (QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) CONSTANT (17.1576 &T29023))
    [37] (QUALITATIVE_BEHAVIOR (VELOCITY SPACESHIP) CONSTANT (17.1576 &T29023))
       [28] (OFF_ROCKETS SPACESHIP (17.1576 &T29023))
        [12] (OFF_ROCKETS SPACESHIP (17.1576 &T29023))
           [10] (STOP-FIRE-ROCKETS SPACESHIP 17.1576)
              [7] (SPACE_VEHICLE SPACESHIP)
              [8] (ON_ROCKETS SPACESHIP (10.0 17.1576))
               {explanation for 6}
              [9] (BEFORE 10.0 17.1576)
       [31] (NEGLIGIBLE (EXTERNAL_FORCE SPACESHIP) (17.1576 &T29023))
        [30] (NEGLIGIBLE (EXTERNAL_FORCE SPACESHIP) (17.1576 &T29023))
           [29] (LOCATION SPACESHIP DEEP-SPACE)
            {explanation for 20}
   [42] (SUBINTERVAL (&TS &TG) (17.1576 &T29023))
    [41] (SUBINTERVAL (&TS &TG) (17.1576 &T29023))
       [39] (ON_AFTER &TS 17.1576)
       [40] (ON_BEFORE &TG &T29023)


General explanation for (MOVING ?SS9056 ?V19035 (?ST19062 ?ET19064)):
[44] (MOVING ?SS9056 ?V19035 (?ST19062 ?ET19064))
   [26] (VALUE (VELOCITY ?SS9056) ?V19035 !T9036)
    [25] (VALUE (VELOCITY ?SS9056) ?V19035 !T9036)
       [13] (VALUE (VELOCITY ?SS9056) ?V09030 ?T19017)
       [14] (CONTINUOUS (VELOCITY ?SS9056))
        [17] (-> ((QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) INCREASING (?T19017
T+INF))) ((VALUE (VELOCITY ?SS9056) ?VMAX9046 T+INF)))
        [16] (-> ((QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) INCREASING (?T19017
T+INF))) ((VALUE (VELOCITY ?SS9056) ?VMAX9046 T+INF)))
           [15] (MAX_VAL (VELOCITY ?SS9056) ?VMAX9046)
       [18] (BETWEEN ?V19035 ?V09030 ?VMAX9046)
       [24] (QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) INCREASING (?T19017 !T9036))
        [23] (QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) INCREASING (?T19017
!T9036))
           [19] (ON_ROCKETS ?SS9056 (?T19017 !T9036))
            [6] (ON_ROCKETS ?SS9056 (?T19017 !T9036))
               [4] (FIRE-ROCKETS ?SS9056 ?T19017)
                  [1] (SPACE_VEHICLE ?SS9056)
                  [2] (OFF_ROCKETS ?SS9056 (?T09018 ?T19017))
                  [3] (BEFORE ?T09018 ?T19017)
           [22] (NEGLIGIBLE (EXTERNAL_FORCE ?SS9056) (?T19017 !T9036))
            [21] (NEGLIGIBLE (EXTERNAL_FORCE ?SS9056) (?T19017 !T9036))
```

```
                              [20] (LOCATION ?SS9056 DEEP-SPACE)
        [38] (QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) CONSTANT (!T9036 &T29023))
          [37] (QUALITATIVE_BEHAVIOR (VELOCITY ?SS9056) CONSTANT (!T9036 &T29023))
              [28] (OFF_ROCKETS ?SS9056 (!T9036 &T29023))
                [12] (OFF_ROCKETS ?SS9056 (!T9036 &T29023))
                    [10] (STOP-FIRE-ROCKETS ?SS9056 !T9036)
                        [7] (SPACE_VEHICLE ?SS9056)
                        [8] (ON_ROCKETS ?SS9056 (?T19017 !T9036))
                          {explanation for 6}
                        [9] (BEFORE ?T19017 !T9036)
              [31] (NEGLIGIBLE (EXTERNAL_FORCE ?SS9056) (!T9036 &T29023))
                [30] (NEGLIGIBLE (EXTERNAL_FORCE ?SS9056) (!T9036 &T29023))
                    [29] (LOCATION ?SS9056 DEEP-SPACE)
                      {explanation for 20}
        [42] (SUBINTERVAL (?ST19062 ?ET19064) (!T9036 &T29023))
          [41] (SUBINTERVAL (?ST19062 ?ET19064) (!T9036 &T29023))
              [39] (ON_AFTER ?ST19062 !T9036)
              [40] (ON_BEFORE ?ET19064 &T29023)

NIL
> (first *plans*)
{PLAN 1}
  [     (VALUE (VELOCITY ?SS9056) ?V09030 ?T19017)
        (CONTINUOUS (VELOCITY ?SS9056))
        (MAX_VAL (VELOCITY ?SS9056) ?VMAX9046)
        (BETWEEN ?V19035 ?V09030 ?VMAX9046)
        (SPACE_VEHICLE ?SS9056)
        (OFF_ROCKETS ?SS9056 (?T09018 ?T19017))
        (BEFORE ?T09018 ?T19017)
        (BEFORE ?T19017 !T9036)
        (LOCATION ?SS9056 DEEP-SPACE)
        (ON_AFTER ?ST19062 !T9036)
        (ON_BEFORE ?ET19064 &T29023)
        (OBSERVABLE (VELOCITY ?SS9056))
    -> (MOVING ?SS9056 ?V19035 (?ST19062 ?ET19064))
    by (FIRE-ROCKETS ?SS9056 ?T19017)
        (STOP-FIRE-ROCKETS ?SS9056 !T9036)
        [MONITOR for (VALUE (VELOCITY ?SS9056) ?V19035 !T9036)]]
```

;;; CLASSICAL PLANNING MODE

```
> (clear-all)

Cache cleared
Working memory cleared
Nodes forgotten
Interval constraints removed
NIL
> (load-nonreactive)
;;; Loading source file "ar-nr2.lisp"

Rules loaded
;;; Loading source file "ae-nr2.lisp"

Example loaded
NIL
> *goal*
(MOVING SHIPSHAPE 100.0 (&TS &TG))
```

```
;;; The goal is to have the spaceship constantly moving at 100 m/s.

> (look-init-data)

Initial Data
     (SPACE_VEHICLE SHIPSHAPE)
     (LOCATION SHIPSHAPE DEEP-SPACE)
     (MASS SHIPSHAPE 13000.0 10.0)
     (SPECIFIED-VALUE (BURN-RATE (FUEL SHIPSHAPE)) -200.0)
     (SPECIFIED-VALUE (VELOCITY (EXHAUST SHIPSHAPE)) -300.0)
     (CONSTANT_COMBUSTION SHIPSHAPE)
     (CONSTANT_EXHAUST SHIPSHAPE)
     (VALUE (VELOCITY SHIPSHAPE) 65.0 10.0)
     (OFF_ROCKETS SHIPSHAPE (-100.0 &T0))
     (AFTER &TS 10.0)
     (AFTER &TG 10.0)
NIL
;;; The spaceship is initially moving at 65 m/s at time 10.

> (run *goal* :wm t :full t)


Initializing working memory...

Attempting to learn plan(s) for (MOVING SHIPSHAPE 100.0 (&TS &TG))

;;; As the system observes every action, it attempts to construct an explanation for the action—i.e
;;;     verify that the action was executable.

Explaining actions...

     Observed (FIRE-ROCKETS SHIPSHAPE 10.0)
     Finding explanations...
     Found 1 explanation(s)

[5] (FIRE-ROCKETS SHIPSHAPE 10.0)
  [4] (FIRE-ROCKETS SHIPSHAPE 10.0)
       [1] (SPACE_VEHICLE SHIPSHAPE)
       [2] (OFF_ROCKETS SHIPSHAPE (-100.0 10.0))
       [3] (BEFORE -100.0 10.0)

     Observed (STOP-FIRE-ROCKETS SHIPSHAPE 17.1576)
     Finding explanations...
     Found 1 explanation(s)

[11] (STOP-FIRE-ROCKETS SHIPSHAPE 17.1576)
  [10] (STOP-FIRE-ROCKETS SHIPSHAPE 17.1576)
       [7] (SPACE_VEHICLE SHIPSHAPE)
       [8] (ON_ROCKETS SHIPSHAPE (10.0 17.1576))
         [6] (ON_ROCKETS SHIPSHAPE (10.0 17.1576))
              [4] (FIRE-ROCKETS SHIPSHAPE 10.0)
                   [1] (SPACE_VEHICLE SHIPSHAPE)
                   [2] (OFF_ROCKETS SHIPSHAPE (-100.0 10.0))
                   [3] (BEFORE -100.0 10.0)
       [9] (BEFORE 10.0 17.1576)

All actions explained.  Attempting to explain goal (MOVING SHIPSHAPE 100.0 (&TS &TG))

Constructed 1 explanation(s) (*explanations*)
```

```
Extracting plan(s)...
   Determining plan components from [EXPLANATION for (MOVING ?VEH9153 ?VF9133 (?ST19155
?ET19157))]

Extracted 1 plan(s)      <Hit <CR> to continue...>


;;; The general plan is checked for conjectured variables. Since this is a classical plan, none will
;;;      be found.

Looking for non-operational components
Plan already operational

Constructed 1 plan(s) (*plans*)
NIL
> (show-explanation (first *explanations*))



Specific explanation for (MOVING SHIPSHAPE 100.0 (&TS &TG)):
[60] (MOVING SHIPSHAPE 100.0 (&TS &TG))
     [46] (VALUE (VELOCITY SHIPSHAPE) 100.0 17.157686)
       [45] (VALUE (VELOCITY SHIPSHAPE) 100.0 17.157686)
           [13] (VALUE (VELOCITY SHIPSHAPE) 65.0 10.0)
           [16] (NEGLIGIBLE (EXTERNAL_FORCE SHIPSHAPE) (10.0 17.157686))
             [15] (NEGLIGIBLE (EXTERNAL_FORCE SHIPSHAPE) (10.0 17.157686))
                 [14] (LOCATION SHIPSHAPE DEEP-SPACE)
           [44] (LINEAR_MOMENTUM_CONSERVATION SHIPSHAPE 65.0 10.0 100.0 17.157686)
             [43] (LINEAR_MOMENTUM_CONSERVATION SHIPSHAPE 65.0 10.0 100.0 17.157686)
                 [31] (DETERMINE_PARAMETERS SHIPSHAPE 65.0 10.0 100.0 17.157686 -300.0
-200.0 13000.0)
                   [30] (DETERMINE_PARAMETERS SHIPSHAPE 65.0 10.0 100.0 17.157686 -300.0
-200.0 13000.0)
                         [22] (CHANGING (VELOCITY (EXHAUST SHIPSHAPE)) -300.0 (10.0
17.157686))
                           [21] (CHANGING (VELOCITY (EXHAUST SHIPSHAPE)) -300.0 (10.0
17.157686))
                               [17] (SPACE_VEHICLE SHIPSHAPE)
                               [18] (ON_ROCKETS SHIPSHAPE (10.0 17.157686))
                                 [6] (ON_ROCKETS SHIPSHAPE (10.0 17.157686))
                                     [4] (FIRE-ROCKETS SHIPSHAPE 10.0)
                                         [1] (SPACE_VEHICLE SHIPSHAPE)
                                         [2] (OFF_ROCKETS SHIPSHAPE (-100.0 10.0))
                                         [3] (BEFORE -100.0 10.0)
                               [19] (CONSTANT_EXHAUST SHIPSHAPE)
                               [20] (SPECIFIED-VALUE (VELOCITY (EXHAUST SHIPSHAPE))
-300.0)
                         [28] (CHANGING (BURN-RATE (FUEL SHIPSHAPE)) -200.0 (10.0
17.157686))
                           [27] (CHANGING (BURN-RATE (FUEL SHIPSHAPE)) -200.0 (10.0
17.157686))
                               [23] (SPACE_VEHICLE SHIPSHAPE)
                                 {explanation for 17}
                               [24] (ON_ROCKETS SHIPSHAPE (10.0 17.157686))
                                 {explanation for 6}
                               [25] (CONSTANT_COMBUSTION SHIPSHAPE)
                               [26] (SPECIFIED-VALUE (BURN-RATE (FUEL SHIPSHAPE)) -200.0)
                         [29] (MASS SHIPSHAPE 13000.0 10.0)
           [42] (COMPUTE-TF 65.0 10.0 100.0 13000.0 -200.0 -300.0 17.157686)
            [41] (COMPUTE-TF 65.0 10.0 100.0 13000.0 -200.0 -300.0 17.157686)
               [39] (COMPUTE-DT 65.0 10.0 100.0 13000.0 -200.0 -300.0 7.1576858)
                [38] (COMPUTE-DT 65.0 10.0 100.0 13000.0 -200.0 -300.0 7.1576858)
                   [32] (DIFF 100.0 65.0 35.0)
                   [33] (QUOT 35.0 -300.0 -0.11666667)
                   [34] (E -0.11666667 0.8898818)
                   [35] (PROD 13000.0 0.8898818 11568.463)
                   [36] (DIFF 11568.463 13000.0 -1431.5371)
                   [37] (QUOT -1431.5371 -200.0 7.1576858)
               [40] (SUM 10.0 7.1576858 17.157686)
   [55] (QUALITATIVE_BEHAVIOR (VELOCITY SHIPSHAPE) CONSTANT (17.157686 &T29077))
    [54] (QUALITATIVE_BEHAVIOR (VELOCITY SHIPSHAPE) CONSTANT (17.157686
&T29077))
       [49] (NEGLIGIBLE (EXTERNAL_FORCE SHIPSHAPE) (17.157686 &T29077))
        [48] (NEGLIGIBLE (EXTERNAL_FORCE SHIPSHAPE) (17.157686 &T29077))
           [47] (LOCATION SHIPSHAPE DEEP-SPACE)
            {explanation for 14}
       [53] (NEGLIGIBLE (INTERNAL_FORCE SHIPSHAPE) (17.157686 &T29077))
        [52] (NEGLIGIBLE (INTERNAL_FORCE SHIPSHAPE) (17.157686 &T29077))
           [50] (SPACE_VEHICLE SHIPSHAPE)
            {explanation for 17}
           [51] (OFF_ROCKETS SHIPSHAPE (17.157686 &T29077))
            [12] (OFF_ROCKETS SHIPSHAPE (17.1576 &T29077))
               [10] (STOP-FIRE-ROCKETS SHIPSHAPE 17.1576)
                  [7] (SPACE_VEHICLE SHIPSHAPE)
                  [8] (ON_ROCKETS SHIPSHAPE (10.0 17.1576))
                   {explanation for 6}
                  [9] (BEFORE 10.0 17.1576)
   [59] (SUBINTERVAL (&TS &TG) (17.157686 &T29077))
    [58] (SUBINTERVAL (&TS &TG) (17.157686 &T29077))
       [56] (ON_AFTER &TS 17.157686)
       [57] (ON_BEFORE &TG &T29077)


General explanation for (MOVING ?VEH9153 ?VF9133 (?ST19155 ?ET19157)):
[60] (MOVING ?VEH9153 ?VF9133 (?ST19155 ?ET19157))
   [46] (VALUE (VELOCITY ?VEH9153) ?VF9133 &T29073)
    [45] (VALUE (VELOCITY ?VEH9153) ?VF9133 &T29073)
       [13] (VALUE (VELOCITY ?VEH9153) ?VI9134 ?TI9144)
       [16] (NEGLIGIBLE (EXTERNAL_FORCE ?VEH9153) (?TI9144 &T29073))
        [15] (NEGLIGIBLE (EXTERNAL_FORCE ?VEH9153) (?TI9144 &T29073))
           [14] (LOCATION ?VEH9153 DEEP-SPACE)
       [44] (LINEAR_MOMENTUM_CONSERVATION ?VEH9153 ?VI9134 ?TI9144 ?VF9133
&T29073)
        [43] (LINEAR_MOMENTUM_CONSERVATION ?VEH9153 ?VI9134 ?TI9144 ?VF9133
&T29073)
           [31] (DETERMINE_PARAMETERS ?VEH9153 ?VI9134 ?TI9144 ?VF9133 &T29073
?VE9136 ?ME9142 ?MI9139)
            [30] (DETERMINE_PARAMETERS ?VEH9153 ?VI9134 ?TI9144 ?VF9133 &T29073
?VE9136 ?ME9142 ?MI9139)
               [22] (CHANGING (VELOCITY (EXHAUST ?VEH9153)) ?VE9136 (?TI9144
&T29073))
                [21] (CHANGING (VELOCITY (EXHAUST ?VEH9153)) ?VE9136 (?TI9144
&T29073))
                   [17] (SPACE_VEHICLE ?VEH9153)
                   [18] (ON_ROCKETS ?VEH9153 (?TI9144 &T29073))
                    [6] (ON_ROCKETS ?VEH9153 (?TI9144 &T29073))
                       [4] (FIRE-ROCKETS ?VEH9153 ?TI9144)
```

```
                          [1] (SPACE_VEHICLE ?VEH9153)
                          [2] (OFF_ROCKETS ?VEH9153 (?T09072 ?TI9144))
                          [3] (BEFORE ?T09072 ?TI9144)
              [19] (CONSTANT_EXHAUST ?VEH9153)
              [20] (SPECIFIED-VALUE (VELOCITY (EXHAUST ?VEH9153)) ?VE9136)
               [28] (CHANGING (BURN-RATE (FUEL ?VEH9153)) ?ME9142 (?TI9144
&T29073))
                [27] (CHANGING (BURN-RATE (FUEL ?VEH9153)) ?ME9142 (?TI9144
&T29073))
                   [23] (SPACE_VEHICLE ?VEH9153)
                    {explanation for 17}
                   [24] (ON_ROCKETS ?VEH9153 (?TI9144 &T29073))
                    {explanation for 6}
                   [25] (CONSTANT_COMBUSTION ?VEH9153)
                   [26] (SPECIFIED-VALUE (BURN-RATE (FUEL ?VEH9153)) ?ME9142)
               [29] (MASS ?VEH9153 ?MI9139 ?TI9144)
           [42] (COMPUTE-TF ?VI9134 ?TI9144 ?VF9133 ?MI9139 ?ME9142 ?VE9136
&T29073)
            [41] (COMPUTE-TF ?VI9134 ?TI9144 ?VF9133 ?MI9139 ?ME9142 ?VE9136
&T29073)
               [39] (COMPUTE-DT ?VI9134 ?TI9144 ?VF9133 ?MI9139 ?ME9142 ?VE9136
?DT9143)
                [38] (COMPUTE-DT ?VI9134 ?TI9144 ?VF9133 ?MI9139 ?ME9142 ?VE9136
?DT9143)
                   [32] (DIFF ?VF9133 ?VI9134 ?ETOP9135)
                   [33] (QUOT ?ETOP9135 ?VE9136 ?EEXP9137)
                   [34] (E ?EEXP9137 ?ETERM9138)
                   [35] (PROD ?MI9139 ?ETERM9138 ?SUB9140)
                   [36] (DIFF ?SUB9140 ?MI9139 ?MTOP9141)
                   [37] (QUOT ?MTOP9141 ?ME9142 ?DT9143)
               [40] (SUM ?TI9144 ?DT9143 &T29073)
   [55] (QUALITATIVE_BEHAVIOR (VELOCITY ?VEH9153) CONSTANT (&T29073 &T29077))
    [54] (QUALITATIVE_BEHAVIOR (VELOCITY ?VEH9153) CONSTANT (&T29073 &T29077))
       [49] (NEGLIGIBLE (EXTERNAL_FORCE ?VEH9153) (&T29073 &T29077))
        [48] (NEGLIGIBLE (EXTERNAL_FORCE ?VEH9153) (&T29073 &T29077))
           [47] (LOCATION ?VEH9153 DEEP-SPACE)
            {explanation for 14}
       [53] (NEGLIGIBLE (INTERNAL_FORCE ?VEH9153) (&T29073 &T29077))
        [52] (NEGLIGIBLE (INTERNAL_FORCE ?VEH9153) (&T29073 &T29077))
           [50] (SPACE_VEHICLE ?VEH9153)
           {explanation for 17}
           [51] (OFF_ROCKETS ?VEH9153 (&T29073 &T29077))
            [12] (OFF_ROCKETS ?VEH9153 (&T29073 &T29077))
               [10] (STOP-FIRE-ROCKETS ?VEH9153 &T29073)
                  [7] (SPACE_VEHICLE ?VEH9153)
                  [8] (ON_ROCKETS ?VEH9153 (?TI9144 &T29073))
                   {explanation for 6}
                  [9] (BEFORE ?TI9144 &T29073)
   [59] (SUBINTERVAL (?ST19155 ?ET19157) (&T29073 &T29077))
    [58] (SUBINTERVAL (?ST19155 ?ET19157) (&T29073 &T29077))
       [56] (ON_AFTER ?ST19155 &T29073)
       [57] (ON_BEFORE ?ET19157 &T29077)

NIL
> (first *plans*)

{PLAN 2}
 [   (VALUE (VELOCITY ?VEH9153) ?VI9134 ?TI9144)
     (CONSTANT_EXHAUST ?VEH9153)
```

```
        (SPECIFIED-VALUE (VELOCITY (EXHAUST ?VEH9153)) ?VE9136)
        (CONSTANT_COMBUSTION ?VEH9153)
        (SPECIFIED-VALUE (BURN-RATE (FUEL ?VEH9153)) ?ME9142)
        (MASS ?VEH9153 ?MI9139 ?TI9144)
        (DIFF ?VF9133 ?VI9134 ?ETOP9135)
        (QUOT ?ETOP9135 ?VE9136 ?EEXP9137)
        (E ?EEXP9137 ?ETERM9138)
        (PROD ?MI9139 ?ETERM9138 ?SUB9140)
        (DIFF ?SUB9140 ?MI9139 ?MTOP9141)
        (QUOT ?MTOP9141 ?ME9142 ?DT9143)
        (SUM ?TI9144 ?DT9143 &T29073)
        (LOCATION ?VEH9153 DEEP-SPACE)
        (SPACE_VEHICLE ?VEH9153)
        (OFF_ROCKETS ?VEH9153 (?T09072 ?TI9144))
        (BEFORE ?T09072 ?TI9144)
        (BEFORE ?TI9144 &T29073)
        (ON_AFTER ?ST19155 &T29073)
        (ON_BEFORE ?ET19157 &T29077)
     -> (MOVING ?VEH9153 ?VF9133 (?ST19155 ?ET19157))
     by (FIRE-ROCKETS ?VEH9153 ?TI9144)
        (STOP-FIRE-ROCKETS ?VEH9153 &T29073)]
  > (dribble)
```

References

[Agre87]	P. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 268–272.
[Allen83]	J. F. Allen and J. A. Koomen, "Planning Using a Temporal World Model," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 741–747.
[Bennett89]	S. W. Bennett, "Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains," M.S. Thesis, ECE, University of Illinois, Urbana, Il., January 1989. (Also appears as Technical Report UILU–ENG–89–2204, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign)
[Birnbaum86]	L. Birnbaum, "Integrated Processing in Planning and Understanding," PhD. Thesis , Yale University, Dept. of Computer Science, New Haven, CT, 1986.
[Blythe89]	J. Blythe and T. M. Mitchell, "On Becoming Reactive," *Proceedings of The Sixth International Workshop on Machine Learning*, June 1989, pp. 255–257.
[Brooks82]	R. A. Brooks, "Symbolic Error Analysis and Robot Planning," Memo 685, MIT AI Lab, Cambridge, MA, September 1982.
[Brooks87]	R. A. Brooks, "Planning is Just a Way of Avoiding Figuring Out What To Do Next," Technical Report 303, MIT AI Lab, Cambridge, MA, 1987.
[Chapman87]	D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence 32*, 3 (1987), pp. 333–378.
[Chien89]	S. A. Chien, "Using and Refining Simplifications: Explanation–based Learning of Plans in Intractable Domains," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 590–595.
[Cohen87]	W. W. Cohen, "A Technique for Generalizing Number in Explanation–Based Learning," ML–TR–19, Department of Computer Science, Rutgers University, New Brunswick, NJ, September 1987.
[Cohen89]	P. R. Cohen, M. L. Greenberg, D. M. Hart and A. E. Howe, "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments," *Artificial Intelligence Magazine 10*, 3 (1989), pp. 32–48.
[Collins87]	G. C. Collins, "Plan Creation: Using Strategies as Blueprints," Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, CT, 1987.
[DeJong86]	G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176. (Also appears as Technical Report UILU–ENG–86–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)
[DeJong89]	G. F. DeJong, "Explanation–Based Learning with Plausible Inferencing," *Proceedings of The Fourth European Working Session on Learning*, Montpellier, Dec. 1989, pp. 1–10.
[de Kleer84]	J. de Kleer and J. S. Brown, "A Qualitative Physics Based on Confluences," *Artificial Intelligence 24*, (1984),.
[Dean88]	T. Dean and M. Boddy, "An Analysis of Time–Dependent Planning," *Proceedings of The Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, August 1988, pp. 49–54.
[Doyle86]	R. Doyle, D. Atkinson and R. Doshi, "Generating Perception Requests and Expectations to Verify the Execution of Plans," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 81–88.
[Durfee88]	E. Durfee and V. Lesser, "Predictability vs. Responsiveness: Coordinating Problem Solvers in Dynamic Domains," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 66–71.

[Ellman88] T. Ellman, "Approximate Theory Formation: An Explanation–Based Approach," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 570–574.

[Fikes71] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence 2*, 3/4 (1971), pp. 189–208.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251–288.

[Firby87] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 202–206.

[Forbus84] K. D. Forbus, "Qualitative Process Theory," *Artificial Intelligence 24*, (1984), pp. 85–168.

[Forbus89] K. D. Forbus, "Introducing Actions into Qualitative Simulation," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 1273–1278.

[Genesereth87] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Palo Alto, CA, 1987.

[Halliday81] D. Halliday and R. Resnick, *Fundamentals of Physics*, John Wiley & Sons, New York, 1981.

[Hammond86] K. Hammond, "Learning to Anticipate and Avoid Planning Failures through the Explanation of Failures," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 556–560.

[Hammond89] K. J. Hammond, "Opportunistic Memory," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 504–510.

[Hayes–Roth79] B. Hayes–Roth and F. Hayes–Roth, "A Cognitive Model of Planning," *Cognitive Science 3*, (1979), pp. 275–310.

[Hogge87] J. C. Hogge, "Compiling Plan Operators from Domains Expressed in Qualitative Process Theory," *Proceedings of The Sixth National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 229–233.

[Iwasaki85] Y. Iwasaki and P. Friedland, "The Concept and Implementation of Skeletal Plans," *Journal of Automated Reasoning 1*, 1 (1985), pp. 161–208.

[Konolige89] K. Konolige and M. E. Pollack, "Ascribing Plans to Agents," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 924–930.

[Kuipers84] B. Kuipers, "Commonsense Reasoning About Causality: Deriving Behavior from Structure," *Artificial Intelligence 24*, (1984), pp. 169–204.

[McCarthy69] J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence,"in *Machine Intelligence 4*, B. Meltzer and D. Michie (ed.), Edinburgh University Press, Edinburgh, Scotland, 1969.

[McDermott82] D. McDermott, "A Temporal Logic for Reasoning About Processes and Plans," *Cognitive Science 6*, 2 (1982), pp. 101–155.

[Minton85] S. Minton, "Selectively Generalizing Plans for Problem–Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, August 1985, pp. 596–599.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar–Cabelli, "Explanation–Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47–80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation–Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551–555. (Also appears as Technical Report UILU–ENG–86–2216, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Mostow89] J. Mostow and A. E. Prieditis, "Discovering Admissible Heuristics by Abstracting and Optimizing: A Transformational Approach," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 701–707.

[Sacerdoti74] E. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence 5*, (1974), pp. 115–135.

[Sacerdoti77] E. Sacerdoti, *A Structure for Plans and Behavior*, American Elsevier, New York, 1977.

[Schoppers87] M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 1039–1046.

[Shavlik87] J. W. Shavlik and G. F. DeJong, "An Explanation–Based Approach to Generalizing Number," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 236–238. (Also appears as Technical Report UILU-ENG–87–2220, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Shekhar89] S. Shekhar and S. Dutta, "Minimizing Response Times in Real Time Planning and Search," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 238–242.

[Shoham86] Y. Shoham, "Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence," PhD. Thesis, Yale University, Dept. of Computer Science, New Haven, CT, 1986.

[Stefik81] M. Stefik, "Planning and Metaplanning (MOLGEN: Part 2)," *Artificial Intelligence 16*, 2 (1981), pp. 141–170.

[Suchman87] L. A. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, 1987.

[Sussman73] G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Report 297, MIT AI Lab, Cambridge, MA, 1973.

[Tennenholtz89] M. Tennenholtz and Y. Moses, "On Cooperation in a Multi–Entity Model," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 918–923.

[Thomas68] G. B. Thomas, *Calculus and Analytic Geometry*, Addison–Wesley, Reading, MA, 1968.

[Turney89] J. Turney and A. Segre, "SEPIA: An Experiment in Integrated Planning and Improvisation," *Proceedings of The American Association for Artificial Intelligence Spring Symposium on Planning and Search*, March 1989, pp. 59–63.

[Wilkins88] D. E. Wilkins, *Practical Planning: Extending the Classical Artificial Intelligence Planning Paradiigm*, Morgan Kaufman, San Mateo, CA, 1988.

[Williams84] B. Williams, "Qualitative Analysis of MOS Circuits," *Artificial Intelligence 24*, (1984),.

[Zlotkin89] G. Zlotkin and J. S. Rosenschein, "Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 912–917.

Index

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-90-1605	2.	3. Recipient's Accession No.
4. Title and Subtitle LEARNING COMPLETABLE REACTIVE PLANS THROUGH ACHIEVABILITY PROOFS			5. Report Date May 1990 6.
7. Author(s) Melinda Tumaneng Gervasio			8. Performing Organization Rept. No. R-90-1605
9. Performing Organization Name and Address Dept. of Computer Science 1304 W. Springfield Ave. Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. N00014 86 K 0309
12. Sponsoring Organization Name and Address Office of Naval Research			13. Type of Report & Period Covered Technical 14.

15. Supplementary Notes

16. Abstracts

This thesis presents an integrated approach to planning wherein a classical planner is augmented with the ability to defer achievable goals and address these deferred goals during execution. This integration gains from reactive planning the ability to utilize runtime information, thus reducing the need for perfect a priori information, while retaining the goal-directedness afforded by a priori planning. This approach also retains the provably-correct nature of plans constructed by a classical planner by requiring that all deferred goals have achievability proofs guaranteeing their eventual achievement. Proving achievability is shown to be possible for certain classes of problems without having to determine the actions to achieve the associated goals. General plans for use in this integrated approach are learned through a modified explanation-based learning strategy called contingent explanation-based learning. In contingent EBL, deferred goals are represented using conjectured variables, which act as placeholders for the eventual values of plan parameters whose values are unknown prior to execution. Completors are incorporated into general plans for the runtime determination of values to replace the conjectured variables. Since only conjectured variables with accompanying achievability proofs are allowed into contingent explanations, the general plans learned in contingent EBL are guaranteed to be completable. An implemented system demonstrates the use of contingent EBL in learning general completable reactive plans, which enables the construction of robust, efficient plans for spaceship acceleration.

17. Key Words and Document Analysis. 17a. Descriptors

classical planning
reactive planning
explanation-based learning
imperfect domain theory problem

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 81
	20. Security Class (This Page UNCLASSIFIED	22. Price