Planning For Conjunctive Goals

by

David Chapman
B.S., Massachusetts Institute of Technology
(1982)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS OF THE
DEGREE OF

MASTER OF SCIENCE
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 1985

Signature of Author

Department of Electrical Engineering and Computer Science
January 22, 1985

Certified by

Charles Rich
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Planning For Conjunctive Goals

by

David Chapman

Submitted to the Department of Electrical Engineering and Computer Science
on January 25, 1985 in partial fulfillment of the
requirements for the Degree of Master of Science in
Electrical Engineering and Computer Science

Abstract

The problem of achieving conjunctive goals has been central to domain-independent planning research; the nonlinear constraint-posting approach has been most successful. Previous planners of this type have been complicated, heuristic, and ill-defined. I present a simple, precise algorithm and prove it correct and complete. I also analyze previous work on domain-independent conjunctive planning; in retrospect it becomes clear that all conjunctive planners, linear and nonlinear, work the same way. I give suggestions for future research, identifying the traditional add/delete-list representation for actions as the crucial limitation on the usefulness of contemporary planners.

Thesis supervisor: Dr. Charles Rich

Title: Principle Reseach Scientist

Contents

Plan to capture baby Roo

by Rabbit

1. *General Remarks.* Kanga runs faster than any of Us, even Me.
2. *More General Remarks.* Kanga never takes her eye off Baby Roo, except when he's safely buttoned up in her pocket.
3. *Therefore.* If we are to capture Baby Roo, we must get a Long Start, because Kanga runs faster than any of Us, even me. (*See* 1.)
4. *A Thought.* If Roo had jumped out of Kanga's pocket and Piglet had jumped in, Kanga wouldn't know the difference, because Piglet is a Very Small Animal.
5. Like Roo.
6. But Kanga would have to be looking the other way first, so as not to see Piglet jumping in.
7. See 2.
8. *Another Thought.* But if Pooh was talking to her very excitedly, she might look the other way for a moment.
9. And then I could run away with Roo.
10. Quickly.
11. *And Kanga wouldn't discover the difference until Afterwards.*

— *Winnie The Pooh*

Chapter 1

Introduction

This thesis describes a planner, TWEAK, that is little different from Austin Tate's NONLIN, or Earl Sacerdoti's NOAH. However, after reading it, you will understand TWEAK better than you could expect to understand NONLIN or NOAH. Planners have previously been described as complicated, heuristic, ill-defined AI programs, without specifying clear conditions under which they work. So what? Deep understanding is important in itself, of course; and it's often necessary to further research. I've found it difficult and time-consuming to understand just how previous planners worked; maybe this thesis will make it easier for others.

The main motivation for TWEAK, though, is that if you intend to use a planner as a workhorse "black-box" part of something else, you care whether it works. I started work on planning because I wanted a planner to co-routine with a learner to make an integrated problem-solver. (This experiment is reported on in [naive].) I'd heard that NOAH was the state of the art in planning, and decided to copy it exactly, since I had no interest in the matter. Four readings of [NOAH] and three misconceived implementations later, I had a planner that worked, but no idea why. To determine whether it would work as a reliable subroutine, I had to simplify the algorithm and representations and to apply some mathematical rigor; and that is what you see here. To quote Sacerdoti:

> [The basic operations of NOAH] ... were developed in an ad hoc fashion. No attempt has been made to justify the transformations that they perform, or to enable them to generate all transformations. However, it should be possible to define an algebra of plan tranformations ... It may be possible to develop a body of formal theory about the ways in which interacting subgoals can be deal with.

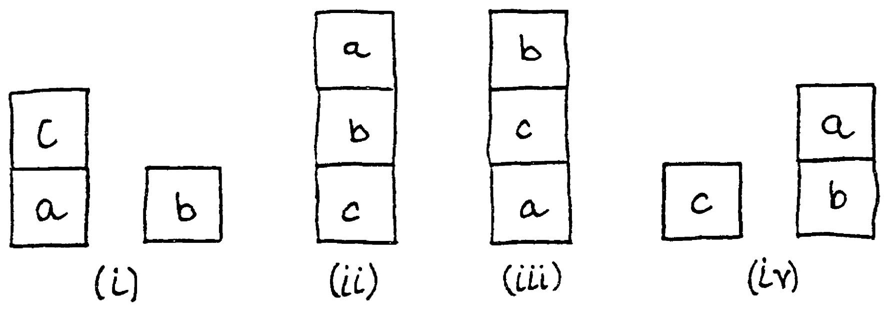
That is what I've done in this thesis.

Roger Schank has made a distinction between "neat" and "scruffy" styles of AI. Planning research to date has been mainly scruffy: heuristic, ill-understood. unclear. Scruffy research is hard to duplicate. It has taken me months of poring over Sacerdoti's book to understand just what he was doing and several misconceived implementations to replicate his results. This is not his fault, or mine: most AI research is necessarily like that. When hacking at the frontiers of knowlege, if you wait to proceed until you understand clearly what you are doing, you will make no progress. Now, ten years after NOAH. I can present TWEAK as a simple, precise algorithm, and prove it correct.

Neat and scruffy research on a particular domain should follow each other in cycles. At the end of this neat thesis. I will make some scruffy suggestions about how to go beyond the crucial limitation of the domain-independent planners that have been implemented to date.

1.1 Non-linear conjunctive planning

The conjunctive planning problem, which has been a main focus of planning research for the last ten years. is to achieve several goals simultaneously: to find a plan that makes a conjunctive formula true after it has been executed. A means of achieving each conjunct separately is taken as a given. Why is conjunctive planning hard? The problem lies in interactions between the means of achieving the individual goals. The following classic problem, known as the "Sussman anomaly", illustrates the difficulty. Suppose that we have three blocks, a, b, and c; initially c is on a and a and b are on the table (situation *i* in the figure). We want to have a stacked on b on c, or to achieve the conjunctive goal (and (on a b) (on b c)) (situation *ii*). Let's say you're only allowed to move one block at a time. If you try to put b on c first, when you go to put a on b you fail, because c is on a and so blocks it from moving (situation *iii*). On the other hand, if you try first to put a on b (removing c to make a accessible,) putting b on c is made impossible by a, which is in the way (situation *iv*).



The Sussman anomaly. (i) the initial situation; (ii) the goal situation; (iii), (iv), the consequences of the two incomplete linear plans.

I'll return to this example later in the thesis and show how non-linear planning can solve it. The important idea. due to Sacerdoti. is that a plan (at least while it is being constructed) does not have to specify fully the order of execution of its steps. In other words, a plan is a partial order on steps, some of which can be unordered.

1.2 Guide to this thesis

The next chapter explains how and why TWEAK works. The exposition is intended to be clear and precise, and so it is written rather in the style of a mathematics text. The chapter is divided into four sections: the first is an informal overview; the second explains what a plan is; the third shows how to manipulate plans; and the fourth describes the overall control structure of the planner.

Chapter three is "related and future work". I analyze previous planning research using the analytical tools developed in chapter two, showing that all domain-independent conjunctive planners work the same way. I suggest that the restrictions on representations of actions that these planners depend upon are their crucial limitation.

The last chapter presents conclusions and suggests an angle of attack upon the problems raised in chapter three.

Chapter 2

TWEAK

TWEAK is a rational reconstruction of previous non-linear planners. This chapter describes the algorithm and proves it correct. TWEAK comes in three layers: a plan representation, a way to make a plan achieve a goal, and a top-level control structure. Each layer is described in more detail in one of the three sections of this chapter.

The plan representation is the most complex layer. The basic operation provided by this representation determines whether a proposition will be true of the world after part of a plan has been executed. An efficient algorithm for this operation depends on a subtle theorem about incompletely defined plans, proved in section 2.1. 2.2 describes a nondeterministic procedure which transforms a plan so that it achieves a goal that it did not previously. The top-level control structure, described in 2.3 controls this nondeterminism. Because this is difficult, a search is used, so that if the wrong choice is made, it is possible to back up.

2.1 The plan representation

In this section I will define plans, problems, and what it means for a plan to solve a problem. I will present an criterion which allows TWEAK to reason about what is true in the world as a plan is executed. Most of the mathematical rigor in the thesis is in this section; accordingly, I've divided it in two: the first part explains all you need to know to understand the rest of the paperthesis the second part proves the criterion correct. Inevitably most of this section is composed of dry and obvious definitions.

TWEAK uses constraint posting in its basic operations. Constraint posting is the definition of an object, a plan in this case, by successively specifying more and more partial descriptions it must fit. Alternatively, constraint posting can be viewed as a search strategy in which rather than generating and testing specific alternatives, chunks of the search space are progressively removed from consideration by constraints that rule them out, until finally every remaining alternative is satisfactory. The advantage of the constraint posting approach is that one does not have to commit oneself to properties of object being searched for before there is information available with which to make a reasoned decision. This often reduces the amount of backtracking necessary.

As TWEAK works on a problem, it has at all times a current plan, which is a partial specification of a plan that may solve the problem. The current plan could be completed in many different ways, depending on what constraints are added to it; thus the current plan represents the class of plans that are its completions. The current plan supplies partial knowledge of the complete plan that will eventually be chosen; ideally all possible completions of the current plan should solve the given problem. I will say "necessarily foo" if foo is true of all possible completions of the current plan, and "possibly foo" if foo is true of some completion of the current plan. The number of completions of a plan is exponential in the number of steps and in the number of variables, so computing whether foo is possible or necessary by searching completions would be very expensive. Much of this section is devoted to describing relatively efficient algorithms that compute possible and necessary properties of an incomplete plan.

A complete plan is a total "time" order on a set of steps, which represent actions. The plan is executed by performing the actions corresponding to the steps in the order given. A step has a set of preconditions, which are things that must be true about the world for it to be possible to take the action. A step also has postconditions, which are things that will be true about the world after the corresponding action has been taken. Pre- and postconditions are both expressed as propositions. Propositions can be positive or negative, and have a content, which is a tuple of elements. Elements can be variables or constants. Functions, propositional operators and quantification are not allowed: all propositions are function-free atomic. (In section 3.2.1 I'll explain why.) Two propositions are negations of each other if one is positive and one is negative and they have the same content.

Plans in TWEAK can be incomplete in two ways: the time order may be incompletely specified, using temporal constraints, and steps may be incompletely specified, using codesignation constraints. A temporal constraint is a requirement that one step be before another; thus a set of temporal constraints is simply a partial order on steps. A completion of a set of temporal constraints C is any total order O on the same set of steps such that sCt implies sOt.

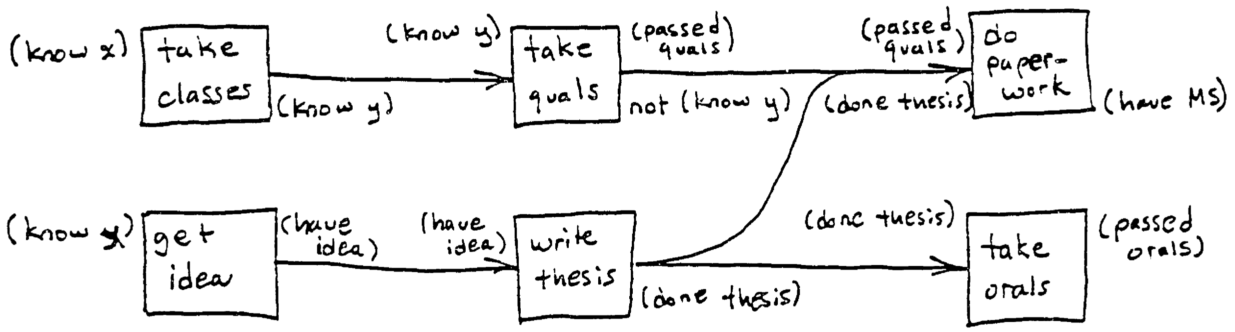
In a complete plan, each variable that appears in a pre- or postcondition must be bound to a specific constant. In execution, that constant will be substituted for the variable when the action is performed. Codesignation is an equivalence relation on variables and constants used to implement binding. Codesignation constraints are requirements of codesignation or non-codesignation of elements; a variable is bound to a constant if the two codesignate. Distinct constants may not codesignate. Two propositions codesignate if both are positive or both are negative and if their contents are of the same length and if corresponding elements in the contents codesignate. For example, the propositions (p a x) and (p a y) codesignate iff x and y codesignate. You can constrain two possibly codesignating propositions to necessarily codesignate by constraining corresponding elements codesignating; this amounts to unification of the two propositions. You can constrain two possibly codesignating propositions necessarily non-codesignating by choosing some index and constraining non-codesignation between the two elements at that index in the content tuples of the two propositions. Codesignation constraints can be maintained using an extension of the union-find algorithm.

TWEAK represents the state of the world with a set of propositions. This set changes as steps are executed. A plan has an initial situation. which is a set of propositions describing the world at the time that the plan is to be executed. Associated with each step in a plan is its input situation, which is the set of propositions that are true in the world just before it is executed. and its output situation, which is the set of propositions that are true in the world just after it is executed. In a complete plan. the input situation of each step is required to be the same set as the output situation of the previous step. The final situation of a plan has the same set of propositions in it as the output situation of the last step. The time order extends to situations: the initial and final situations are before and after every other situation respectively. The input situation of a step is before the step and after every other situation that is before the step; the output situation of a step is after the step and before any other situation that is after the step.

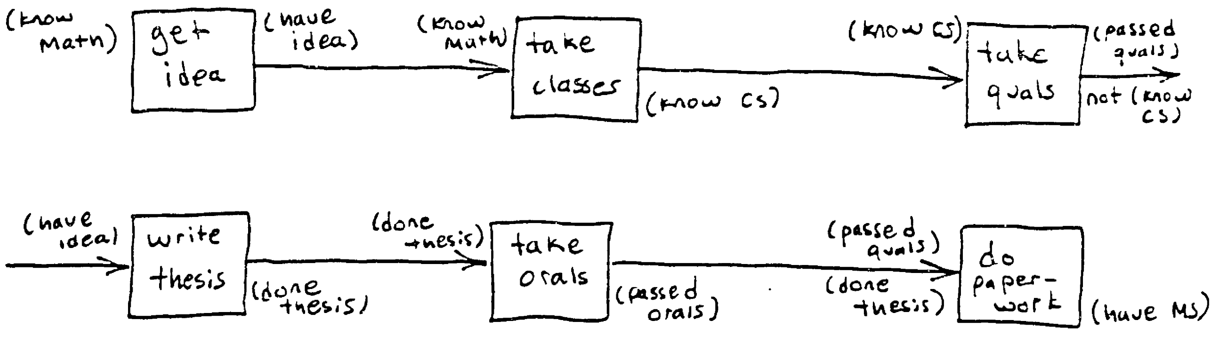
Say that a proposition is true in a situation if it codesignates with a proposition that is a member of the situation. Say that a step asserts a proposition in its output situation if the proposition codesignates with a postcondition of the step. Say that a proposition is asserted in the initial situation if it true in that situation. A proposition is denied in a situation if its negation is asserted there. It's illegal for a proposition to be both asserted and denied in a situation.

A step can be executed only if all its preconditions are in true in its input situation. In this case, the output situation is just the input situation minus any propositions denied the step, plus any propositions asserted by the step. (This is not the same thing as the input situation plus the propositions asserted by the step: if p were true in input situation and the step asserts $\sim p$ (the negation of p), then the output situation must not contain both p and $\sim p$; input and output situations must be consistent sets of propositions, since they describe states of the world.) This model of execution does not allow for indirect or implied effects of actions; any changes in the world must be explicitly mentioned as postconditions.

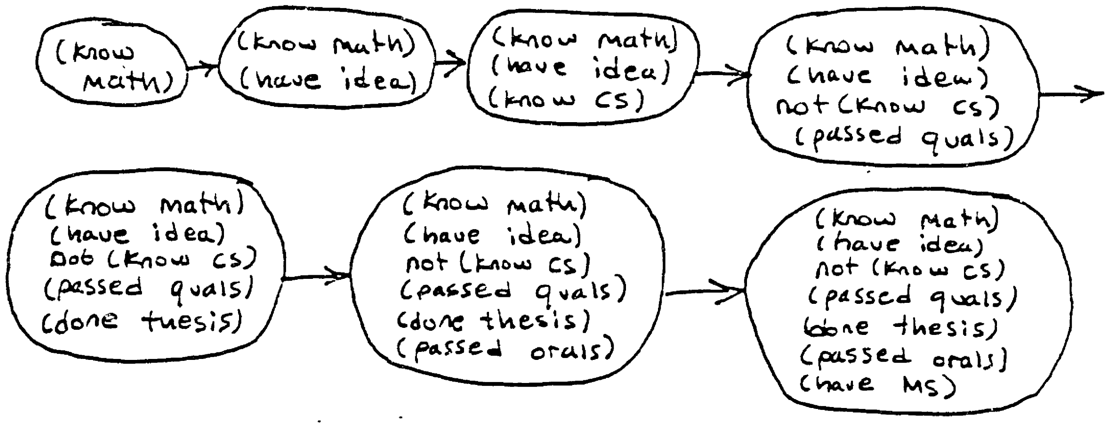
I will use graphs, as in the figure, to illustrate plans. Steps are boxes; the preconditions are put before the box and postconditions after. The steps may have labels inside, but these are only mnemonic. Arcs represent the partial time order.



An incomplete plan.

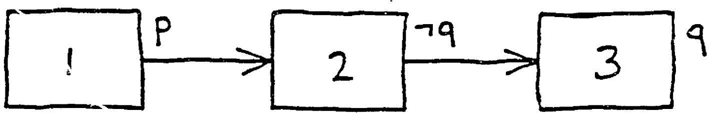


One completion of the example plan.


(know math)
(know math) (have idea)
(know math) (have idea) (know CS)
(know math) (have idea) not (know CS) (passed quals)
(know math) (have idea) not (know CS) (passed quals) (done thesis)
(know math) (have idea) not (know CS) (passed quals) (done thesis) (passed orals)
(know math) (have idea) not (know CS) (passed quals) (done thesis) (passed orals) (have MS)

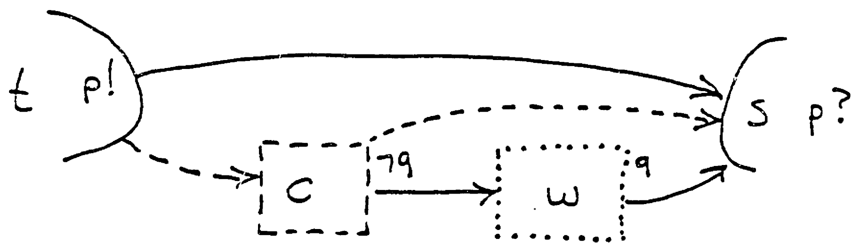
The sequence of situations resulting from execution of the completion.

During planning, incompleteness introduces uncertainty into the meaning of a plan. To use a blocks world example, if v is a variable, after asserting (on block v), there's no way to tell whether (on block v) is true or false, unless v codesignates with a particular constant. I will now sketch the derivation of a criterion that tells you when a proposition is necessarily true in a situation. Certainly a proposition is necessarily true in situation if it is necessarily asserted in it. Once a proposition has been asserted, it remains true until denied. Thus a proposition p is necessarily true in a situation if there is some previous situation in which it is necessarily true, and there is no possibly intervening step that possibly denies it: for if there is a step that is even possibly inbetween that even possibly denies p, we can find a completion in which the step actually is inbetween and actually denies p. (A step possibly denies p by denying a proposition q which possibly codesignates with p). There is an exception to this rule, illustrated by the following odd plan:



If p and q are possibly codesignating. this plan has two classes of completions: one in which p and q actually codesignate. in which case p is asserted by step 3: and one in which p and q are noncodesignating. so that p is asserted by step 1. and is never denied. In either case, p is true in the final situation. even though there is no step that necessarily asserts p without an intervening step possibly denying it. All these observations together suggest the following criterion, which I will prove at the end of this section:

Modal truth criterion: A proposition p is necessarily true in a situation s iff there exists a situation t necessarily equal or previous to s in which p is necessarily asserted and such that there is no step C possibly between t and s which denies a proposition q possibly codesignating with p, unless there is a step necessarily between C and s which asserts r, such that $q \approx p \Rightarrow r \approx p$. The criterion for possible truth is exactly analogous, with all the modalities switched (read "necessary" for "possible" and vice versa).



The necessary truth criterion. Solid lines indicate necessarily time-relatedness and dashed lines possible time-relatedness; the dashed box, a disallowed step; the dotted box a step that would make the dashed step legal.

This criterion can be computed in polynomial time, though it does exponentially much "work" by describing properties of the exponentially large set of completions of a plan. The remainder of TWEAK depends heavily on this theorem; its proof is surprisingly complex.

Now I will define problems and their solutions. A problem is an initial situation and a final situation, which are two sets of propositions. A plan for a problem is one such that every proposition in its initial situation is true in the initial situation of the problem. A goal is a proposition which must be achieved (true) in a certain situation. The goals of a plan for a problem are defined to be the propositions in the final situation of the problem, which must be true in the final situation of the plan, and the preconditions of steps in the plan, which must be true in the corresponding input situations. A complete plan for a problem solves the problem if all its goals are achieved. Thus, a complete plan solves a problem if it can be executed in the initial situation of the problem and if the final situation of the problem is a correct partial description of the world after execution. The aim of TWEAK is to produce a plan that necessarily solves the problem it is given. This plan may be incomplete; in this case any of its completions can be chosen for execution.

Having read this far you know all you need to about plans. The dispensable remainder of this section is devoted to proving the modal truth criterion. I will prove the criterion in three steps. First, I prove the "time's arrow lemma", which says that only the steps executed before a situation are relevant to what is true in that situation. This is used to prove a truth criterion for complete plans that is analogous to the modal truth criterion. I use that to prove the modal truth criterion.

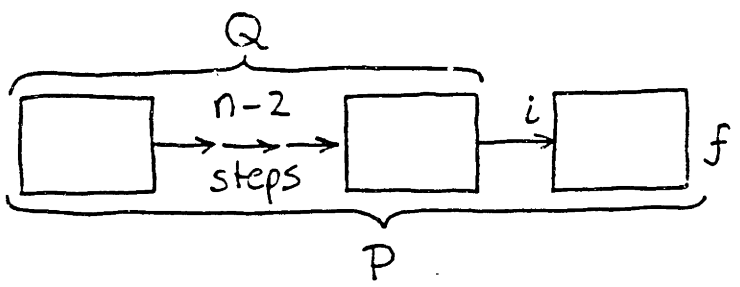
Time's arrow lemma: Let $\mathcal{P}$ and $\mathcal{Q}$ be complete plans whose initial situation and first n steps are identical. p is true in the initial situation or the input or output situation of one of the first n steps iff it is true in the corresponding situation in $\mathcal{Q}$.

Proof: By induction on n. If $\mathcal{P}$ and $\mathcal{Q}$ have no steps, they have only one situation, which is both initial and final, and the same in both. Certainly p is true in this situation in $\mathcal{P}$ iff it is true in the corresponding situation in $\mathcal{Q}$. Suppose now that the lemma is true for plans whose initial situation and first $n-1$ steps are identical; I will show that it holds for plans whose initial situation and first n steps are identical. Let $\mathcal{P}$ and $\mathcal{Q}$ be such plans; certainly then they also have the first $n-1$ steps identical, so by the induction hypothesis p is true in the initial situation and the input and output situations of the first $n-1$ situations of $\mathcal{P}$ iff p is true in the analogous situation of $\mathcal{Q}$. The only remaining situation we need check is the output situation s of the nth step S. By definition, s is the input situation of S minus any propositions denied by S, plus any propositions asserted by S. If p is neither asserted nor denied by S, then it is true in s just in case it is true in the input situation of S, which, by the induction hypothesis, is iff it is true in the input situation of the analogous step of $\mathcal{Q}$. If p is asserted or denied by S, it is also asserted or denied by the analogous step in $\mathcal{Q}$ and so again is true in s iff it is true in the output situation of the nth step of $\mathcal{Q}$. By induction, then, the lemma holds for any n. $\square$

Truth criterion for complete plans: In a complete plan, a proposition p is true in a situation s iff there exists a situation t previous or equal to s in which p is asserted and such that there is no step between t and s which denies p.

Proof: It should be obvious that this criterion is correct. Informally, we start with the initial situation and at each step delete from the set of propositions representing the world what is denied by the step and add what is asserted by it. Everything else is preserved untouched.

The rigorous proof again uses induction on the length of plans. A plan with no steps has only an initial and a final situation, and the two are equal. A proposition is true in the initial situation iff it is equal to something in the initial situation, in which case it is asserted there. A proposition is true in the final situation iff it is true in the initial situation. In both cases, there is no possibility of an intervening denying step.



Constructions used in the proof of the truth criterion for complete plans

Suppose now that the criterion is correct for complete plans of length n; I will show that it is correct for complete plans of length $n+1$. Let $\mathcal{P}$ be a plan of length $n+1$, f the final situation of $\mathcal{P}$, S the last step in $\mathcal{P}$, i be the input situation of S, and $\mathcal{Q}$ the plan of length n derived by removing S from $\mathcal{P}$. p is true in a situation in $\mathcal{P}$ other than f just in case p is true in the corresponding situation of $\mathcal{Q}$, by the time's arrow lemma. Then by the induction hypothesis, the criterion holds for every proposition and every situation in $\mathcal{P}$ except perhaps f (which is equal to the output situation of S). f is by definition i minus things denied by S plus whatever is asserted by S. Thus p is true in f iff it is asserted by S or true in i and not denied by S. In the former case, p is asserted in f; in the latter, by the induction hypothesis there exists a situation t before i in which p is asserted, and there is no step between t and i that denies p. This same t is before f and there is no step between t and f denying p. $\Box$

Modal truth criterion: A proposition p is necessarily true in a situation s iff there exists a situation t necessarily equal or previous to s in which p is necessarily asserted and if there is no step C possibly before s which denies a proposition q possibly equal to p and such that there is no step W necessarily between C and s which asserts r, a proposition such that $q \approx p$ implies $r \approx p$. The criterion for possible truth is exactly analogous, with all the modalities switched (read "necessary" for "possible" and vice versa).

In proving this theorem, I will call situations t necessarily before s that necessarily assert p "winners;" steps C possibly before s that possibly deny p "clobberers," and steps necessarily between a clobberer and s that necessarily assert a proposition whose codesignation with p is implied by the codesignation with p of the proposition the clobberer denies "white knights." A clobberer with a matching white knight is a "foiled clobberer;" one without is a "dastardly clobberer."

The white knight part of the criterion is an embarassing appendix; it would be much more elegant if the modal criterion were identical to the criterion for complete plans, except with some "possibly"s and "necessarily"s thrown in. The white knight clause is needed to make the only-if proof go through; we'll see in the next section that it doesn't actually do anything for you.

Proof: I'll give the proof for necessary truth; possible truth follows by the fact that something is possibly true iff its negation is not necessarily true. The criterion is composed of two independent conjuncts: existence of a winner and absence of a dastardly clobberer. We can distinguish three cases: those in which there is no winner; those in which there is a dastardly clobberer; and those in which there is a winner and no dastardly clobberer. To prove the criterion, which is a statement of equivalence, I need to show that in the first two cases p is not necessarily true in s and that in the third case it is necessarily true.

The first case is that in which there is no winner; in this case I show that p is not necessarily true in s, by exhibiting a completion such that p is not true in s. To construct this completion, first put after s every step possibly after s. For there to exist a completion of the resulting plan, these constraints must not conflict. Ordering constraints can conflict with each other only if they imply $a < b \wedge b < a$ for some a and b. This could happen only if a cyclic chain of orderings were set up. Since each new constraint relates some step to s, this chain would have to pass through s: some step would have to be made both before and after s. The new constraints only put steps after s, so the cycle must run from s to some newly constrained step S to some step that was already necessarily before s. But if S was already necessarily before a step necessarily before s, S was already necessarily before s, contrary to hypothesis.

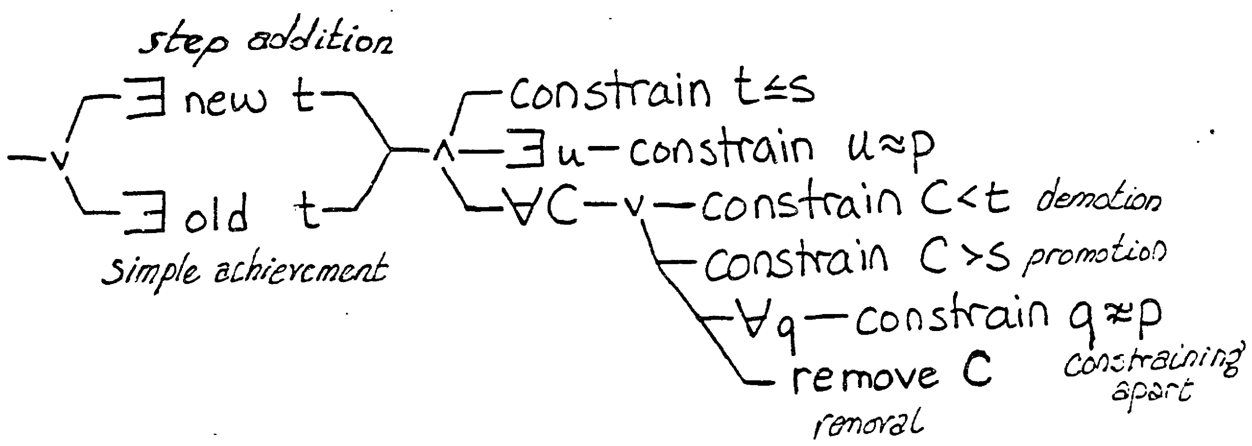
The next step is to constrain every variable that is not already constrained codesignating with some specific constant to codesignate with a distinct "gensymmed" constant that does not appear elsewhere in the plan. Again I need to show that all these constraints can be added without conflict. The constraint $v \approx c$ could cause problems only if $v \not\approx c$ is also necessarily true. Since c is not mentioned anywhere else in the plan, and since v is not already bound, this could not happen.

Now a completion in which p is not true in s is made by taking any completion of this modified plan. Such a completion exists, because there are no conflicts in the constraints. In such a completion there is no step that necessarily asserts p and that is necessarily before s; and any step that only possibly asserts p has been made not to assert it (but rather some other proposition involving gensymmed constants) and any step that necessarily asserts p but is only possibly before s has been put after s. So there is no situation before or equal to s in which p is asserted, and by the truth criterion for complete plans p is not true in s in this completion.

The second case is that in which there is a dastardly clobberer C. Again I will construct a completion such that p is not true in s. This completion is made by putting C before s, then

So to achieve p in s, you must find a situation t satisfying the body of the outermost existential. This t could be either a situation already in the plan or the output situation of a step that you add to the plan. In either case, you must then guarantee the three conjuncts of the body of the existential. The first can be satisfied by constraining t to be before s. To satisfy the second conjunct, you can't change the set of things asserted in t, so you can only constrain an existing asserted proposition to codesignate with p. The third conjunct ensures that no step clobbers p. To satisfy it, you must guarantee the body of the universal for every step C in the plan. There are four ways this can be done: you can constrain every postcondition of C not to codesignate with the negation of p, you can put C before t or after s, or you can simply remove C from the plan. One could imagine also adding a step W asserting q between C and s. This is actually useless: either p will end codesignating with q or not; in the first case one might as well add a step that asserts p and put C before it, and in the second one might as well constrain p and q non-codesignating.

The following diagram describes the process of making a plan achieve a goal:



In this figure, $\vee$ means to choose one of the alternate paths; $\wedge$ tells you to do all the paths; $\exists$ means "choose a"; $\forall$ tells you to apply the following path to every one; the verb phrases tell you to take some action.

Because the modal truth criterion is sufficient as well as necessary, this achievement procedure encompasses *all* the ways to make a plan achieve a goal. In this respect TWEAK can not be improved upon.

Several operations in the procedure are postings of constraints. These constraints may be incompatible with existing constraints: for example, you can't constrain $C < t$ if there is already a constraint that $C > t$. The constraint maintenance mechanism signals failure in such cases, and the top-level control structure backtracks.

The goal-achievement procedure has the useful property that so long as step addition and removal are avoided, the new plan will continue to necessarily achieve any goals that it previously did. That's because the rest of the procedure operates only by adding constraints. When constraints are added, things that were previously possibly true become either necessarily true or necessarily false, but nothing that is necessarily true can change its truth value.

Step addition adds new preconditions to the plan that need to be achieved, and the added step may also deny, and so disachieve, old goals. This is unavoidable, and it can lead to infinite looping. Therefore, TWEAK prefers constraint posting to step addition. TWEAK does not make use of step removal at all. Every step is introduced to assert some goal proposition, and so removing one will make negative progress. It is never the case that the only way to achieve a goal is to remove a step.

Step addition involves choosing what step to add. Every step in a plan must represent an action that is possible to execute in the domain in which the problem is specified. Thus, along with a problem, the user of TWEAK must give a list of possible actions. To achieve a goal p by addition, the added step must assert a proposition possibly codesignating with p. The choice of steps, then, is among those that are allowed in the domain and that possibly assert the desired goal.

2.3 The top-level control structure

TWEAK begins work on a problem with a first plan whose initial situation is the initial situation of the problem and which has no steps or constraints. It then enters a loop in which some goal not yet achieved is chosen and the procedure of the last section is applied, yielding a new plan. When all the plan's goals are achieved, the plan solves the problem. Choosing which goal to achieve and which choices to make in the achievement procedure is very difficult; certainly it is not always possible to choose right the first time. Therefore the top-level control structure of TWEAK is a search through the space of alternate paths through the goal achievement procedure.

It is important to understand that TWEAK's search is in the space of alternate ways to achieve goals, not in the space of plans. The difference is crucial to efficiency. Searching the space of plans is extremely inefficient. TWEAK can do much better, because the achievement procedure does in fact achieve goals, so that progress is relatively quick. In effect, it uses its understanding of the ways that goals can be achieved to speed up search.

People have thought a lot about what sort of search to use; this work is reviewed in section 3.4. Since none of the search strategies developed so far seem very good, I simply use dependency-directed breadth-first search in TWEAK. I shan't argue for breadth-first search; it's certainly too expensive for general use. However, the use of dependency-directed search deserves some justification.

Dependency-directed backtracking [Doyle] is more efficient than chronological backtracking only if the search space is nearly decomposable into independent subparts, so that after a failure in one part, only the work done on that part needs to be undone; work on other parts can be saved. TWEAK does have this property when running in many domains. For example, in the blocks world, if the goal is to build two disjoint structures, the search space can be divided into the part concerned with building the one and the part concerned with building the other. Failure in building the one structure will not affect partial successes achieved thus far in building the other.

Because the step addition can make the plan grow arbitrarily large, the search may never converge on a correct plan. In fact, there are three possible outcomes: success, in which a correct plan is found; failure, when the planner has exhaustively searched the space of sequences of plan modification operations, and every branch fails; and nontermination, when the plan grows larger and larger and more and more operations are applied to it, but it never converges to solve the problem.

Lemma: Each of the three outcomes is possible for some choice of problem.

Proof: A trivial example of success is a problem with a single goal which is true in the initial situation. A trivial example of failure is provided by a problem that has at least one goal that is not true in the initial situation and which is not possibly asserted by any available action. An example of non-termination is given by the problem whose initial state is $\sim g$ and $\sim h$ and whose goals are g and h with a conjunct planner that for goal g returns a single-step plan with precondition $\sim h$ and postcondition g and for goal h returns a step with precondition $\sim g$ and postcondition h. TWEAK loops on this problem, building plans that are longer and longer chains of steps that alternately assert g and h. □

This is the central theorem of this thesis:

Correctness/completeness theorem: If TWEAK, given a problem, terminates claiming a solution, the plan it produces does in fact solve the problem. If TWEAK returns signalling failure or does not halt, there is no solution to the problem.

Proof: This follows directly from the use of the necessary truth criterion in computing the plan-solves-problem? predicate and in constructing the goal achievement procedure. TWEAK's current plan always has the same initial situation as the problem given, and the top-level loop continues until all the problem goals and all the subgoals are achieved, at which point the plan must solve the problem. If there is a solution to the problem, it must be a plan that in some way achieves the problem's goals; and TWEAK examines all possible ways to do so. The plan must also have all preconditions satisfied; but TWEAK also tries all possible ways to satisfy preconditions. Thus, if a solution exists, TWEAK will find it. □

Chapter 3

Past and future planning research

This chapter is "related and future work". It is much longer than such sections are in typical AI papers because conjunctive planning is an unusual subfield of AI in showing a clear line of researchers duplicating and building on each other's work. Science is supposed to be like that, but for the most part AI hasn't been. The two main points of the chapter are that in retrospect all conjunctive planners work the same way and that the action representation which they depend on is inadequate for real-world planning.

I will restrict attention to domain-independent conjunctive planning, ignoring planners and parts thereof that are domain-dependent or non-conjunctive. This may seem unfair at times. There are two previous survey articles on this topic, [Sacerdoti-tactics] and [Tate-expert]. The facts I will consider are much the same as those covered by the other papers; my analyses of many points are different. The principal contribution of TWEAK is the rigor of my formulation of non-linear constraint-posting planning. There is a series of three papers, [WARPLAN], [Waldinger], [Rosenschein], building on each other, that treat linear planning rigorously, and prove correctness of linear planners. Those papers were motivated by many of the same considerations as this one: rigor requires simplicity, guarantees agreement about details, can unveil problems and suggest solutions. Thus these papers form the neat part of the scruffy-neat research cycle for linear planning. The neat part of the cycle for nonlinear planning begins with this thesis.

The first section in this chapter is a historical overview of domain-independent conjunctive planning, showing how different planners build on one another, with particular emphasis on the history of the ideas embodied in TWEAK. The other three sections are devoted to the three levels of a conjunctive planner: representation, plan modification operations, and top-level search strategies. The most interesting suggestions for future research are in section 3.2.1 on action representation; the most interesting analysis of past work is in section 3.3 on plan modification operations.

3.1 Chronology

There are two important "prehistorical" non-conjunctive planners that introduced techniques that underlie all the conjunctive planning work. GPS [GPS], due to Allen Newell, J.C. Shaw, and Herbert Simon, introduced means-ends analysis, which is to say step addition or subgoaling: solving problems by applying an operator that would achieve some goal of the problem, and taking the preconditions of the operator as new goals. STRIPS [STRIPS], due to Richard Fikes and Nils Nilsson, contributed the action model—in which steps have postconditions which are the only things that get changed by the step—that is used by all domain-independent conjunctive planners.

Domain-independent conjunctive planning begins in 1973 with Sussman's HACKER [HACKER]. This is ironic, in that HACKER was intended to be a learning program, more than a planner; Sussman happened to apply learning to planning as a domain. HACKER used many techniques, described later, that have never been duplicated. Sussman ended his thesis with the problem described in section 1.1, due to Allen Brown but widely known as "the Sussman anomaly," which HACKER could not solve without resort to what Sussman called a "hack".

The urge to find a clean solution to the Sussman anomaly drove a series of rapid developments over the next four years. David Warren's WARPLAN [WARPLAN] and Austin Tate's INTERPLAN [INTERPLAN-memo], [INTERPLAN-IJCAI], [INTERPLAN-thesis], both of 1974, cleaned up Sussman's ad-hoc "hack": declobbering by promotion, in fact. Richard Waldinger [Waldinger] further generalized promotion.

In 1975 came Sacerdoti's NOAH [NOAH], [Sacerdoti-plans], the first nonlinear planner. Besides his improvement in the representation of plans, Sacerdoti substantially expanded the set of plan modification operations. Tate (the same author of INTERPLAN) improved on NOAH in 1976. NONLIN [NONLIN-TR], [NONLIN-IJCAI] had a backtracking top-level control structure, so that it could find plans after NOAH would get stuck, and added to NOAH's set of plan modification operations.

After 1976, there was a great drought for many years. During this period, there was one important piece of work on non-conjunctive planning: Mark Stefik's 1980 MOLGEN [MOLGEN] made constraints a central planning issue for the first time. Conjunctive planning was not advanced until a new spurt of work beginning in 1982.

All the new conjunctive planners were NOAH-based. Several researchers extended NOAH by improving the representation of time, in quite different ways. (These improvements have not been incorporated in TWEAK.) Vere's DEVISER [DEVISER] treated actions as temporal intervals with numerical endpoints. James Allen and Johannes Koomen's planner [Allen] also treated actions as intervals, but was based on Allen's non-numerical time logic. Drew McDermott [McDermott-time] suggested using a time logic based on branching futures as a basis for planning. David Wilkin's SIPE [SIPE-AIJ], [SIPE-IJCAI] used MOLGEN-like constraints and had a new technique for detecting clobbering.

3.2 Representation

A planner must represent time, actions the agent can take, and the world and the objects in it. Domain-independent planners all base their representations on those of STRIPS, and with the exception of the introduction of constraints, have not progressed much beyond that framework. Therefore, this section will be more concerned with future than with past work.

The rest of this section is divided into four decreasingly interesting subsections. The first discusses action representation; the second time; the third binding constraints; and the fourth is a grab-bag miscellany section.

3.2.1 Actions

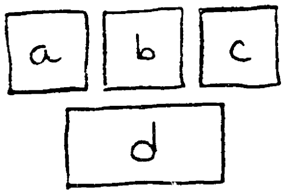
I think the most important area of future domain-independent planning research is the extension of understood techniques to more expressive action representations. The TWEAK action representation requires that all changes made by an action be explicitly represented as postconditions; many actions can not be formalized in this representation. The effects of some actions depend on the situation in which they are applied: for example, a push-push toggle switch turns a light on if it is off, and off if it is on. TWEAK can not represent this action, because no constant set of postconditions expresses its behavior. Other actions have indirect effects: if block b is on block a and we move a from $room_1$ to $room_2$, b will also move. Again this can not be expressed.

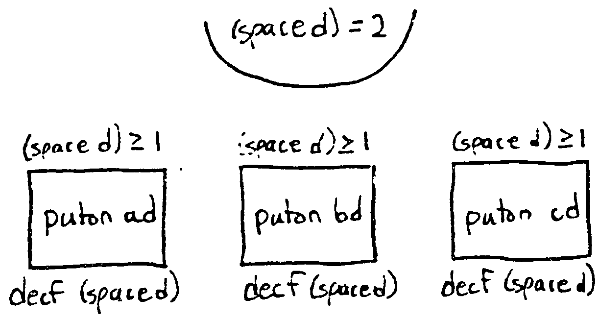
Why is TWEAK's action representation thus impoverished? Because the modal truth criterion depends on this simplicity. The "frame problem" [Raphael], [JanLert] is to find efficiently-implementable truth criteria; TWEAK and similar planners solve the frame problem by so restricting the action representation that my very simple truth criterion holds. There are conjunctive planners that do not make such restrictions, but to the extent that they do not, their truth criteria are incorrect, and the planners themselves are only heuristic.

The restrictions on action representation make TWEAK almost useless as a real-world planner. It is barely possible to formalize the cubical blocks world in the TWEAK representation; HACKER's blocks world, with different sized blocks, is impossible to formalize. Thus, I think the foremost challenge of future planning research is to find truth criteria that will allow conjunctive domain-independent planning to apply to real-world problems; I'd like to see a black-box planner as part of every expert system in ten years.

I will use the rest of this subsection to show how my modal truth criterion can fail with more expressive action representations. In the conclusion of this thesis, chapter 4, I'll suggest a direction of research for finding truth criteria that might cope with such extensions.

Consider a blocks world in which we will allow zero, one, or two blocks to be on any given block. (This example and its analysis are due to David McAllester, personal communication.) Every block still must be on zero or one other block. We need a function space that takes a block as an argument and returns an integer between zero and two inclusive that tells how much room is left on top of the block. A precondition of `(puton a b)` will be that `(space b)` be greater than zero, and the corresponding postcondition will be that `(space b)` be one less than before. Note that this postcondition can not be expressed in the traditional action representation, because it depends on the input situation. Consider a plan with three unordered steps: `(puton a d)`, `(puton b d)`, and `(puton c d)`, and suppose that space d is two in the initial situation.





A precondition of each step is that space d be at least one. Let us ask whether this precondition p is satisfied in the input situation i of (puton a d). According to the modal truth criterion, it is satisfied so long as there is a situation t (the initial situation will do nicely) necessarily before i in which p is necessarily asserted (all true) and that there is not even possibly a step between t and i that denies p. Candidate clobbering steps are (puton b d) and (puton c d); they are possibly between t and i. Does one deny p? No, because in i there is space for two, and each step only decrements the space by one. Yet the two steps synergistically act together to clobber p. This possibility is not accounted for in the modal truth criterion. Of course, we could try all possible orderings of the three steps and see if p holds in i in every case; but this exponential computation amounts to returning to linear planning, and all the advantages of constraint-posting are lost.

Suppose we allowed indirect effects of actions. These indirect effects would have to be derived by deduction from the direct effects, explicitly represented as postconditions. Call the set of all propositions that follow from another set of propostitions the deductive closure of the latter set. The semantics of executing a step in a situation will be to negate all the propositions in the deductive closure of the postconditions and remove that set from the input situation, add to the result the postconditions, and take the deductive closure of all that. It is again possible for two steps to act synergistically to assert or deny a proposition: if $q \wedge r \Rightarrow p$ and one asserts q and the other r, together they assert p (equivalently deny $\sim p$).

This is the reason that TWEAK requires all propositions to be atomic. Non-atomic propositions could be used, but would be simply treated as literals; the logical operators can't get their usual semantics without deduction.

3.2.2 Time

The representation of time is crucial to planning: a plan is really a representation of part of the future. The biggest advance in domain-independent conjunctive planning was probably the recognition that the time order can be partial, at least until execution. This observation first appears in print in [WARPLAN] p. 16, but the first implementation was in NOAH.

I have simplified the representation of plans from those used in NOAH and NONLIN. Those planners represent plans as directed acyclic graphs in which there were many different types of nodes, only a few of which represent anything much. My plans are simply partial orders on actions.

Much of the post-drought planning research in the last few years has focused on overlapping actions. All the old planners assume that actions are instantaneous and atomic; in the real world most actions take time, and several can happen at once. McDermott's planner of [McDermott-time], DEVISER, and Allen's planner are based on the NOAH/NONLIN framework, but can represent and reason about actions that have durations and that can be executed in parallel. It would be interesting to analyze these planners in the same way I have analyzed TWEAK: particularly, to find a provable truth criterion that accounts for overlapping actions and to see what plan modification operations it engenders.

Allen's time logic can represent the constraint that two actions be disjoint in time without committing to which order they are to be performed in. This might make it possible to defer the choice of declobbering operation further than can be done in TWEAK, since one could combine promotion and demotion into a single constraint, not committing oneself until later as to which is to be used. This decrease in commitment might result in less search. In general, one might represent time propositionally, allowing for instance general disjunctions between several possible constraints to be expressed. This would trade off commitment against the cost of deducing facts about a particular incomplete plan.

Plans are like programs in many ways; but programs have conditionals, iterations, and dataflows, which domain-independent planners have not for the most part been able to generate. Rosenschein's planner [Rosenschein] generates conditionals, but only by randomly adding a branch on a random condition to the plan, which is not very helpful. NOAH had a feature for representing simple iterations; however, this representation does not allow declobbering between steps inside the loop and steps outside, and so can not be called conjunctive. The Programmer's Apprentice [Rich-TR], [Rich-representation] uses a "plan calculus" historically derived in part from NOAH, which can represent conditionals, loops, and dataflow. No program synthesizer has yet been written using this representation.

3.2.3 Binding constraints

MOLGEN was the first planner to highlight the use of constraints; its author, Stefik, introduced the term "constraint posting". Constraints in MOLGEN are arbitrary predicates possibly on several variables. MOLGEN performs three operations on constraints: formulation, propagation, and establishment. Formulation is making new constraints, propagation creates new from old constraints, and establishment is binding variables to values. MOLGEN was the first planner to do propagation; unfortunately his propagation techniques are domain-dependent and not even described in his thesis. Stefik describes a "build or buy decision" in constraint establishment: either one can bind a variable to a constant already appearing in the plan ("buy"), or one can bind it to a new constant. In this case it is often necessary to introduce new steps whose postconditions involve the constant, so as to guarantee properties of it; this is the "build" option. MOLGEN was first to introduce new steps to satisfy a constraint.

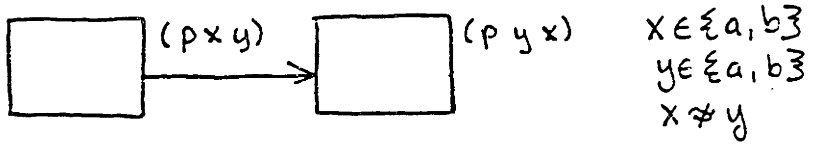
It is little recognized that HACKER used binding constraints. They were implemented as special type preconditions on "formal objects" (variables). HACKER's clever techniques for constraint establishment make use of the CONNIVER [CONNIVER] context mechanism and have not been duplicated since. However, only the "buy" option was considered, and it is not clear how general the implementation was.

INTERPLAN and NONLIN don't have binding constraints. WARPLAN effectively inherits them from PROLOG. It is unclear how variables work in NOAH: they seem to be inherited at least partly from QA4, the implementation language. NOAH has a special node-type in plans called a SOME node which lists the possible bindings of a variable. NOAH seems not to use binding constraints beyond this.

SIPE's constraints are modelled on MOLGEN's. SIPE's truth criterion takes into account possible truths resulting from possibly equal propositions, but does so only heuristically; TWEAK was first to get this right. SIPE, like TWEAK, propagates constraints only via equalities.

TWEAK uses only equality constraints, because preconditions already can represent predicates on variables, so that there is little loss in expressive power. If one looks at the way constraints are used in planning, almost all constraints correspond very naturally to preconditions of steps. There are some exceptions to this in MOLGEN, all of them constraints that have been created via propagation. Stefik's build or buy decision translates in the TWEAK framework into step addition versus simple achievement. Since preconditions are associated with times, predicates on formal objects are also; this solves problems MOLGEN had with time representation.

The difference in expresive power between MOLGEN or SIPE and TWEAK is that TWEAK can not restrict the range of a variable to a finite set. There are two reasons I haven't put range restrictions into TWEAK: because constraint satisfaction then becomes NP-complete (proof by reduction from graph coloring); and more seriously because the truth criterion will fail if I allow them. This "pathological" plan illustrates the problem:



Here the binding constraints require that either x be bound to a and y to b, or vice versa; either way p = (p a b) holds in the final situation. Yet p is not necessarily asserted by any particular step.

There is a "deep" reason variables are needed in the blocks world, in which they originated. In [naive] I describe a problem solver that uses a TWEAK-like planner as a subroutine. This problem solver views puton as both a POP and a PUSH. It is the PUSH aspect that is exploited in achieving on goals, and the POP aspect that is exploited in achieving cleartop goals. When puton is viewed as a POP, there is no explanation for what the second argument (the place to put the block moved) is for. So the problem solver uses a variable to leave the second argument unspecified. Whatever value the second argument takes on, the puton will act as a POP.

3.2.4 Other problems for future research

Representations of the world using non-assertional datastructures are simpler, more efficient, and better reflect its structure than the assertional databases used in current domain-independent planners. Many domain-specific planners use such representations effectively, and I see no inherent difficulty in using such simulation structures in linear domain-independent planning. In constraint-posting planning the state of the world is not completely defined at all times; this is easy to implement using assertional databases, in which it is easy to represent unknown truth values. It is much harder to represent partial knowlege with non-assertional datastructures. A hybrid approach, using both assertions and more direct representations, may lead to simpler, more powerful and efficient planners.

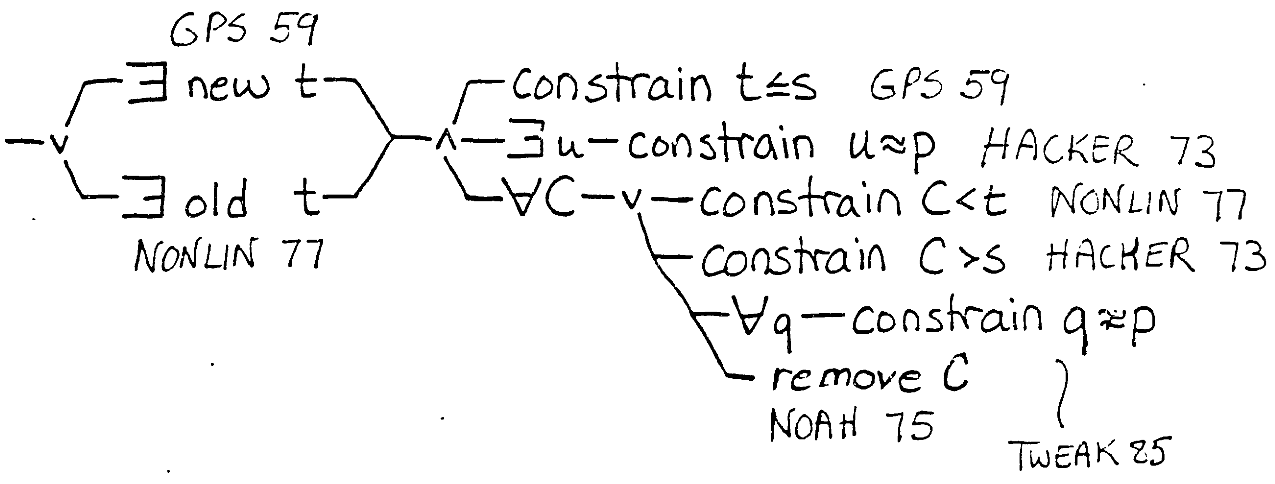
An important problem in planning that is almost entirely open is that of coping with events that are not planned for. These might be the actions of other agents or spontaneous law-governed physical happenings. DEVISER and Allen's planner can plan around "scheduled" unplanned events: these are specific events that will occur at known times. Many problem solvers have plan executives that, when unexpected events occur, call the planner to derive a new plan from the altered state of the world. In the real world, which has tigers in it, that isn't good enough; you have to prepare for contingencies during planning. Such plans need conditionals.

Related problems are planning with incomplete or incorrect knowlege of the world and with unreliable primitive actions. Another area of current research is in planning for multiple agents or multiple effectors that have to be synchronized [Rosenschein-multi-agent], [Georgeff], [Fikes-directions].

3.3 Plan modification operations

In this section, I will treat first linear planners and then non-linear planners. That isn't quite the chronological order, as some linear planners postdate NOAH. There are two interesting points to the section: one is the way the individual plan modification operations were developed by generalization, splitting, and merging. The other is to see that all conjunctive domain-independent planners work in substantially the same way, though they look very different, using apparently unrelated datastructures and algorithms. As time went by, features were added and alternative implementations were tried, but the fundamentals are unchanged from HACKER down to TWEAK. This has not been generally realized, even by the people who wrote the planners. Forcing all the algorithms into the vocabulary of TWEAK modification operations makes them easy to compare. However, many of the early planning papers are very difficult to read, and some of what follows may be inaccurate in detail.

This diagram summarizes the section, illustrating which parts of the achievement procedure were invented when:



The most important thing to understand about linear planners, which has not been appreciated before, is that they work just the same way as non-linear ones, except that the representation is awkward. The basic operation of all the linear planners is analogous to declobbering by promotion and demotion. In a non-linear planner these just add temporal constraints; in a linear planner, a step must be picked up and moved to a different position in the plan.

There are two different versions of promotion that appear in linear planners. The first, which I will call individual promotion, moves the clobberer forward over the clobberee. (Alternatively, the clobberee could be moved backward before the clobberer; this is an uninterestingly different operation.) In a non-linear planner, promotion automatically also puts everything before the clobberee before the clobberer and vice versa; individual promotion doesn't generally do so, with the result that a step can be separated from the steps that were to achieve its preconditions, so that they must be reachieved. The other version of promotion I'll call block promotion: it moves the clobberer, together with the steps that achieve its preconditions, and the steps that achieve their preconditions, transitively, as a block. This implies a strong linearity assumption: not only that the plan can be totally ordered, but also that if you have goals g and h and S achieves g and T achieves h and S is before T, then all the steps that achieve preconditions of S will be before all the steps that achieve preconditions of T. In other words, the time order must respect the subgoaling hierarchy. Using only block promotion, it is impossible to solve optimally the Sussman anomaly problem. The optimal solution involves three steps: (puton c table), (puton b c), (puton a b). This plan violates the strong linearity assumption: the first step achieves the precondition (cleartop a) for the last step, but the middle step is not achieving a precondition of the last, but rather one of the top-level goals.

HACKER has, in effect, four plan modification operations. Step addition is used initially on each of the conjunct goals, and the resulting steps are arbitrarily linearly ordered. HACKER recognizes four bug types, each of which has a corresponding plan modification operation. "Prerequisite Missing" is a precondition that is not true anywhere before it is needed, and is patched with step addition. "Prerequisite Clobbers Brother Goal" is just clobbering, and block promotion is applied. "Prerequisite Conflict Brothers" is a "doublecross": a pair of steps each of which clobbers the other. HACKER has a plan modification operation for this which does not appear in any other planner: a RESOLVE expert is called, which will replace the two steps with a single step that achieves the prerequisites which the two steps were intended to achieve. In practice, it seems that the only cases the RESOLVE expert could handle were pairs of steps that achieved the goals (spacefor *a c*) and (spacefor *b c*) (Sussman's blocks world allows more than one block on a given block). The expert would replace the two steps with a subplan that achieved (spacefor (both a b) *c*).

HACKER had many nifty planning techniques that somehow got lost in the sands of time. For example, HACKER's addition operation is different from those of all subsequent planners. Addition in later planners uses one of the possibly several steps that could achieve a goal, perhaps saving the others as backtrack alternatives. HACKER instead puts all the alternative steps into the plan, arbitrarily linearly ordered. This leads to the fourth bug type, "Strategy Conflict Brothers," in which a step in one strategy (alternative achieving step) clobbers something in another strategy. In this case, HACKER applies promotion. This "multiple addition" operation has many interesting properties. The principal use of it is in achieving (spacefor *a b*); the two strategies are "compacting" the blocks on top of *b* and "punting" blocks off of *b* that don't need to be there. Although either of these strategies may achieve an unachieved spacefor goal, neither is guaranteed to. Yet, if executed in the right order (punt then compact), they will make space if it is possible to do so. This sort of synergy and partial goal fulfillment has never been duplicated.

Sussman called Allen Brown's problem "anomalous" because it could not be solved using block promotion. He presents a solution using individual promotion, but regards this as a "hack." Why? Sussman viewed HACKER as an automatic programming system, constructing programs, not plans. A conjunction (of the original goals or of the preconditions to a step) is achieved via a single subroutine. Promotion in HACKER was confined to permuting the order of lines of a subroutine; this amounts to block promotion, since subroutines encapsulate the subgoal hierarchy. In order to solve the Sussman anomaly, one must move program steps across subroutine boundaries, which HACKER wouldn't do.

It's a pity, though, that the view of planning as automatic programming got lost in the shuffle. HACKER's performance in any given domain would improve as time went by, because the programs it wrote could be re-used on new problems. Sussman describes techniques for generalization and subroutinization of programs so that less planning would need to be done later. Compilation can be viewed as constant-folding the source code into the interpreter; HACKER in effect constant-folded classes of problems into the planner. It would be interesting to build a problem solver that constant-folded into TWEAK, and incorporated what has been learned in the past ten years about generalization.

WARPLAN has two plan modification operations, step addition and an operation that combines addition with individual promotion. The latter operation ("regression") is to find a step that will achieve the goal, then to search backward from the end for a place in the plan where the step can be put without being clobbered. WARPLAN was able to solve the anomalous problem because it doesn't make the strong linearity assumption; it represents plans as flat orders, without hierarchy.

Waldinger [Waldinger] generalizes Warren's technique. Rather than regressing a specific action through the plan until it can be inserted, he regresses the goal to be achieved, and at every step modifies the goal. The idea is that if we want to achieve g after S and can't, to find some h that can be achieved before S so that after S g will be achieved. h will be the same as g under the assumption that actions have constant atomic postconditions; thus Waldinger generalized Warren's technique, correct for a simple action representation, to extended action representations. Waldinger's ideas were not implemented.

INTERPLAN has three plan modification operations: step addition and both versions of promotion (not combined with addition). HACKER is able to discover bugs only by (simulated) execution. WARPLAN never introduces clobberings; the plan was always correct. INTERPLAN and most subsequent planners are able to discover clobberings during planning.

Rosenschein [Rosenschein] describes a linear planner that uses both promotion and demotion He also has an operation for introducing conditionals (if-then-else branches) into plans: it chooses an arbitrary proposition not provable or disprovable from the initial situation, and puts a branch on this proposition at the beginning of the plan. Since Rosenschein's top-level control structure is backtracking search, this technique will eventually try every conditional plan, but since the proposition is chosen at random, it is hardly efficient.

With NOAH comes the great explosion in the set of plan modification operations. NOAH classifies clobberings into three sets: in the first, two steps each clobber the other (a "double cross"); in the second, the clobberer and clobberee are unordered; and in the third, the clobberer is before the clobberee. The case (an "n-cross") in which there is a set of more than two steps that clobber each other, arranged in a cycle, is neglected. I don't understand Sacerdoti's explanation of the plan modification operation to patch double crosses; it seems to be a version of step addition, possibly combined with declobbering by constraining apart. The other two cases are handled by promotion and demotion with addition, respectively. Sacerdoti's description of his promotion operation is inconsistent; he gives a description of a correct operation, but also describes one based on a datastructure called a Table of Multiple Effects which is incorrect (as noticed by Tate).

"Eliminate Redundant Preconditions" is a step removal operation. Step removal is useful because NOAH does not have a simple achievement operation. Thus, if two steps have the same precondition, they may both be achieved by addition, and then one of the two steps removed. "Use Existing Objects" binds variables to constants. Sacerdoti is very unclear on when it is applied and how the binding is chosen. NOAH has a simple but entirely adequate technique for achieving a disjunctive goal. Each of the disjuncts is planned for, until it is clear that one can be achieved; then an operation is applied that removes the incomplete plans for the other disjuncts.

Sacerdoti presents two "task specific" plan modification operations. "Tool Gathering" optimizes plans relative to a notion of the cost of performing correct plans: a correct plan may be made into a better, still correct one by some re-orderings. "Limitations of an Apprentice" compensates for the inexpressibility in the STRIPS of many kinds of actions. The example he gives is very similar (a resource conflict, requiring a global view to do declobbering) to the blocks world example I analyze in section ??. Unfortunately, the details of the operation are not given.

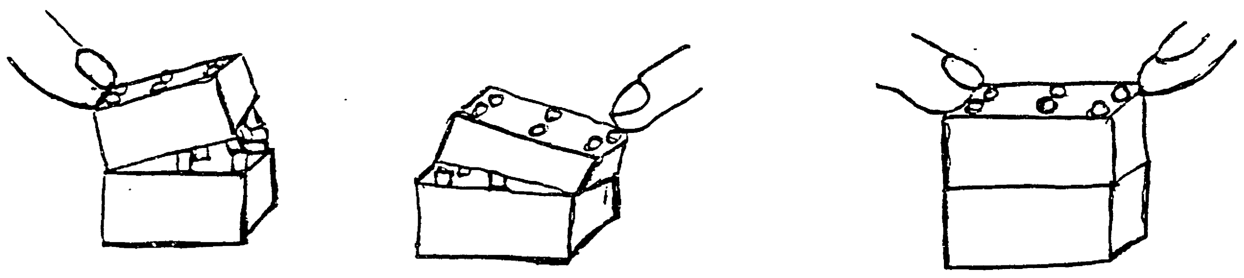
NONLIN was the first planner to use simple achievement; this is defined as in TWEAK, but without handling constraints. NONLIN also uses addition, promotion, and demotion.

SIPE introduced no new modification operations, but it does have a new technique for detecting clobbering. A particularly common sort of precondition is what Wilkins terms a *resource* : a binary variable that must be set to one value ("available") for an operation to be applicable, and which will be set to a different value during the operation, then "released" or reset at the end. Two unordered steps that try to use the same resource clobber each other; SIPE then applies temporal declobbering. The techniques SIPE uses for resource clobbering detection are only heuristic; more work is needed to understand this maneuver.

There's one plan modification operation that may be new to TWEAK: declobbering by constraining apart. I suspect that NOAH's double-cross removal operation, which I don't understand, may combine step addition with declobbering by constraining apart; apart from this, there seems to be no precedent.

Because the modal truth criterion is sufficent as well as necessary, there are no more plan modification operations possible without extending the range of represented actions. Once that is done, new operations will be possible; again they can be derived from the truth criteria for the new representations. For example, the pathological plan on page 25 suggests an operation of "achievement by constraining apart": if the plan did not have the binding constraint $x \not\approx y$, adding this constraint would achieve p in the final situation.

If the representation of time is extended to allow overlapping actions, a new operation ("simultaneous achievment") is possible. In this figure,



someone is trying to stick two LEGO blocks together. Pushing the top block down on either end will tend to make the block pivot around the center pair of posts, so that the two blocks do not mate. The problem can be solved by applying both operations simultaneously, pushing with a finger at each end. (Another solution is to push in the middle, but that's not very interesting.)

Kristian Hammond describes [WOK] WOK, a planner which although domain-specific and non-conjunctive has interesting things to say about goal interactions (the class of effects that includes clobbering). WOK satisfies goals by introducing interactions of known sorts. This is the antithesis of the linear strategy: rather than assuming as a first approximation that goals don't interact, synergistic interactions are used as a basic tool for achieving goals. It is unclear how this approach can be applied to conjunctive planning, but it may be a useful line of future research.

3.4 The top-level control structure

The top-level control structure of almost every domain-independent conjunctive planner is search. Search control is the aspect of domain-independent conjunctive planning that is understood least. Most of the domains to which domain-independent conjunctive planning has been applied have been forgiving: if more than one plan modification operation is applicable to a clobbering or unestablished goal, any of the possibilities will probably do. Thus, it hasn't been necessary to devote a lot of thought to which to choose. However, just about everyone is convinced that in real domains the choices are critical, and a lot of schemes have been proposed for making them. Since none of these have been adequately tested, little is known about which is best.

Almost every planner has a distinct control structure. I've loosely grouped them in seven classes, ordered roughly by the complexity of the backtracking algorithm. The classes are no backtracking, explicitly represented alternatives, dependency-directed modification, chronological backtracking, dependency-directed backtracking, heuristic search, and metaplanning. Many of the planners I discuss actually fit into several of these classes.

The simplest control structure avoids backtracking altogether. Plan modification operations are applied in a fixed order according to fixed criteria until a correct plan is found or it is no longer possible to apply operations. This is not as bad as it sounds, because you can usually make a good guess as to which modification operation to apply: usually, one should prefer simple achievement to step addition, for example. HACKER uses this approach. NOAH comes very close; it backs up only from alternative choices of variable bindings in SOME nodes (see page 25). That NOAH solved many difficult problems shows that the choice of control structure is unimportant in many domains.

A very simple solution to the problem of which modification operation to apply is to choose all possible ones, splitting the plan into several explicitly represented copies. No planner fits altogether in this class as it is very inefficient in general. If some additional principle decides whether for a given choice to use this technique or to use search, the splitting technique may be useful. SIPE and a planning framework described by Hayes-Roth et al. [Hayes-Roth] take this approach.

The simplest backtracking scheme is chronological: when a choice has to be made, one is chosen by some means and the others are saved away. If the plan can not be extended to a solution by further modification, failure is signalled. The most recent choice point is backed up to, and an alternative for the choice is used. When no choices remain, the next most recent choice point is backed up to, and so on. WARPLAN, INTERPLAN, and NONLIN use chronological backtracking.

Chronological backtracking results in the exploration of more blind alleys than necessary. Dependency directed backtracking backs up at failure not to the most recent choice point, but to one responsible for the failure. For a discussion of dependency-directed backtracking in general, see [Stallman-and-Sussman] and [Doyle-TMS]. The first planner to use dependency-direction was Hayes's 1975 route planner [Hayes], which was not conjunctive. Hayes used backtracking to recover only from execution error, rather than from planning error (dead ends) as does TWEAK, although he explicitly considered the latter possibility. Thus his implemented control structure can be termed "dependency-directed modification" rather than backtracking. Hayes's conception of dependency-directed backtracking predates and seems to be independent of its discovery by Stallman and Sussman, to whom it is usually credited.

Although Tate suggests [NONLIN-memo] using dependency-directed backtracking, the first domain-independent conjunctive planner to implement it was that presented by deKleer et al. [explicit]. This linear planner is not otherwise interesting. Phil London's planner [London-planner] represents plans and world states using a TMS, the utility underlying dependency-directed backtracking, but apparently does not use the TMS for backtracking. To do so, the decisions taken in choosing one rather than another plan modification operation—the metaplan—must be represented, and London did not do so. TWEAK is the first non-linear dependency-directed planner.

Heuristic search uses some numerical estimate of "goodness" to decide which order to try choices in. Tate's INTERPLAN and NONLIN use heuristics to control their chronological search. Since making a wrong choice can result in searching a large dead-end subtree, it would be nice to eliminate wrong choices without having to explore their consequences. Kibler and Morris [Kibler] present a control scheme based on negative search heuristics that prune obviously bad choices. However, these heuristics are domain-specific for the blocks world. Siklossy and Roach [ref] use a similar strategy. Corkill [Corkill] describes a NOAH-like planner in which control is distributed among several message-passing processors.

All the control structures discussed so far (with the possible exception of heuristic search) are "syntactic": they don't depend on the specifics of the plan being constructed, but blindly apply some simple algorithm for choosing among alternatives without considering what those alternatives are. Since control of planning is very hard, such methods may be inherently weak. If planning is hard, perhaps we should apply the full power of a problem solver to choosing what to do next. This is the "metaplanning" approach. There is an increasing literature on this ([Batali], [Davis], [Doyle-SEAN], [Stefik-metaplanning] [Wilensky]) most of which is quite vague and none of which applies specifically to domain-independent conjunctive planning. I'll discuss just two metaplanning systems. Doyle's unimplemented SEAN uses (another copy of) the same planner to do metaplanning as to do planning about the domain. The metaplanner in turn is controlled by an identical metametaplanner and so on; Doyle discusses ways to implement this apparently infinite regress.

MOLGEN has only one level of metaplanning, and the metaplanner is quite unlike the domain-level planner. The domain-level planner creates plans for MOLGEN's domain, genetics experiment planning. It has operations that are analogous to the plan modification operations of TWEAK. These operations are selected by a metaplanner which chooses among plan modification operations. The metaplanner is quite simple; it's perhaps grandiose to call it a planner at all.

The idea of using a copy of TWEAK as a metaplanner is attractive: the plan modification operations can be thought of as having well-defined preconditions (that the constraints they impose not conflict with the existing ones, or that a suitable step exists to achieve a goal in the case of addition) and postconditions (the insertion of the new constraint or step). Unfortunately, TWEAK's action representation is too weak to represent the plan modification operations.

Planners can be classified along a dimension orthogonal to search strategy, that of technique used for recover from execution failures. This isn't part of planning proper, but many systems interleave planning with (simulated or actual) execution so that effectively a non-backtracking planner performs search, failing during execution rather than planning, and then returning to the planner to obtain a new plan to recover. HACKER and BUILD use this approach. NOAH proper doesn't, but its planner is connected to an execution system that will re-invoke the planner after execution failure, so that the system as a whole can be put in this class. HACKER and BUILD make use of CONNIVER techniques similar to dependency-direction in order to figure out which planning decision was responsible for the failure and to try another alternative in the choice.

Planning and AI language design have strongly influenced each other. Many of the planners that do search inherit their search discipline from the language they were written in, and many AI languages were designed to make writing the top-level control of planners easier. PLANNER [PLANNER-TR], [PLANNER-IJCAI] was intended as a language for writting planners in; it was the first to supply backtracking automatically. HACKER and BUILD were written in and depend heavily on the abilities of CONNIVER [CONNIVER], a language written in reaction to the difficulties with chronological backtracking in PLANNER. WARPLAN inherits its search from PROLOG. The planner of deKleer et al. was the first program written in AMORD, and inherits its dependency-directed backtracking from AMORD's TMS [AMORD]. TWEAK, too, inherits its dependency-directed backtracking from Dependency-Directed Lisp, a language specifically designed for TWEAK. DDL looks like ordinary Lisp but has an implicit dependency-directed backtracking control structure. It will be described in a forthcoming paper.

Chapter 4

Conclusions

I promised a scruffy ending to this thesis. In the last chapter I argued that solving the frame problem for more expressive action representations is the outstanding problem in domain-independent planning. I doubt that a neat general solution will be found soon; the frame problem is very hard. I have examined a number of specific domains, and found that for all of them it was easy to find a truth criterion, but that these criteria were quite different. Perhaps then we should give up on domain-independent planning: the user of a planner must specify, together with the problem and the set of available actions, a truth criterion to be used.

But perhaps we can do better, at the expense of some scruffiness. In other work [cliches], I have been developing a theory of "intermediate" techniques, which are neither completely general, nor completely domain-specific. The main idea is that there are "cognitive cliches", commonly occuring formal structures in the world, that have attached to them "intermediate competence" that is specific not to a domain, but to a cliche. Intermediate competence is applied by identifying instances of the associated cliche in the world. Thus, a cliche-based system is domain-independent, as any cliche may show up in any domain, yet has to know specific things about the domain it is running in. I envision a cliche-based constraint-posting planner for extended action representations which would have truth criteria specific to cliches that operators in the world might instantiate. A planner with truth criteria for a few dozen cliches might well cover most of the domains likely to be encountered.

An example cliche is *resource* . A resource (I use the term differently than does Wilkins) consists of a state variable in the world which holds a quantity in some total order, together with at least one *consumer* operator, which decreases, relative to the order, the value of the state variable, and at least one *dependent* operator, which has as a precondition that the state variable have a value greater than some threshold. There may also be *producer* operators that increase the value of the state variable. Resources are found in many domains; puton in the HACKER domain is both a consumer and a dependent for the space on any given block. Associated with the resource cliche is a partial truth criterion: the value of the state variable in a situation s is no less than its value in situation t necessarily before s, minus the sum of the amount of decrements due to consumers possibly between t and s, plus the sum of the amount of increments due to producers necessarily between t and s. From this truth criterion we can derive three patching operations: the precondition of a dependent operator can be achieved, if it is not already, by adding producers between t and s, by constraining consumers possibly between t and s to be before t or after s. or by constraining the amounts of consumption or production to be respectively small or large. In the HACKER blocks world, adding producers of space amounts to the "punting" strategy (see page 28), and increasing the amount they increase space by suggests the "compacting" strategy.

Perhaps the most important contribution of this thesis is the introduction of the notion of a provably correct modal truth criterion. Yet I wonder about the psychological reality of such criteria. Anecdotal evidence suggests that humans plan by doing something easy and debugging the result when it fails. Sussman's HACKER worked that way; unfortunately the set of bugs that it could patch are ones that TWEAK never introduces, and so his specific debugging techniques are of no use. It may be that the only solutions to the frame problem we can devise are heuristic; then theories of plan debugging will become very important.

Acknowlegements

My intellectual debt to the great lineage of AI planning researchers is enormous and obvious.

This thesis incorporates suggestions from many readers which made me reformulate over and over again what it is that I was doing. Steve Bagley, Alan Bawden, Mike Brady, Gary Drescher, Margaret Fleck, Walter Hamscher, Scott Layson, David McAllester, Kent Pitman, Charles Rich, Mark Shirley, and Dan Weld contributed much.

Ken Forbus convinced me that my understanding of non-linear planning would make a Master's thesis.

Ed Giniger teaches me biology and compares stories about idiotic lab politics.

My office mate David McAllester put up with my randomness and is the ideal person to bounce math ideas off of.

Jim Vanesse got me through hard times.

My supervisor Chuck Rich supported me through six thesis topic changes and believed in me when I didn't. His ability to debug me when wedged was vital.

Without Hazel Rovno, *this* thesis would never have been written.

References

[Allen] Allen, James F., and Koomen, Johannes A., "Planning Using a Temporal World Model." *IJCAI-83.*

[Batali] Batali, John, "Computational Introspection." MIT AI Memo 701, February, 1983.

[naive] Chapman, David, "Naive Problem Solving and Naive Mathematics." MIT AI Working Paper 249, June, 1983.

[cliches] Chapman, David, "Cognitive Cliches." Forthcoming.

[Corkill] Corkill, Daniel D., "Hierarchical Planning in a Distributed Environment." *IJCAI-79*, pp. 168–175.

[Davis] Davis, Randall, "Meta-Rules: Reasoning about Control." *Artificial Intelligence* 15 (1980), 179–222.

[explicit] deKleer, Johan, Doyle, Jon, Steele, Guy L. Jr., and Sussman, Gerald Jay, "Explicit Control of Reasoning." *ACM SIGPLAN Notices* Vol. 12, No. 8/*ACM SIGART Newsletter* No. 64, combined special issue, proceedings of the Symposium on Artificial Intelligence and Programming Languages, August 1977, pp. 116-125. Also MIT AI Memo No. 427, June 1977.

[AMORD] deKleer, Johan, Doyle, Jon, Rich, Charles, Steele, Guy L. Jr., and Sussman, Gerald Jay, "AMORD, a Deductive Procedure System." MIT AI Memo 435, January, 1978.

[Doyle-TMS] Doyle, Jon, *Truth Maintenance Systems For Problem Solving.* MIT AI Technical Report 419, January 1978.

[Doyle-SEAN] Doyle, Jon, *A Model for Deliberation, Action, and Introspection.* MIT AI Technical Report 581, Cambridge Mass., May, 1980.

[BUILD] Fahlman, Scott Elliott, "A Planning System for Robot Construction Tasks." *Artificial Intelligence* 5 (1974) pp. 1–49.

[STRIPS] Fikes, Richard E., and Nilsson, Nils J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving." *Artificial Intelligence* 2 (1971), pp. 198–208.

[Fikes-directions] Fikes, Richard E., Hart, P.E., and Nilsson, Nils J., "Some New Directions in Robot Problem Solving." Chapter 23 in *Machine Intelligence 7*, Meltzer and Mitchie, eds., Edinburgh University Press, Edinburgh 1972.

[Georgeff] Georgeff, Michael, "A Theory of Action for MultiAgent Planning." *AAAI-84*, pp. 121–125.

[WOK] Hammond, Kristian J., "Planning and Goal Interaction: The use of past solutions in present situations." *AAAI-83*, pp. 148–151.

[Hayes] Hayes, Philip J., "A Representation For Robot Plans." *4th IJCAI.*

[Hayes-Roth] Hayes-Roth, Barbara, Hayes-Roth, Frederic, Rosenschein, Stan, and Cammarata, Stephanie, "Modelling Planning as an Incremental, Opportunistic Process." *IJCAI-79*, pp. 375–383.

[PLANNER-TR] Hewitt, Carl, *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot.* MIT AI Technical Report 258, April, 1972.

[PLANNER-IJCAI] Hewitt, Carl, "Procedural Embedding of Knowlege in PLANNER." *IJCAI-71*, pp. 167–182.

[JanLert] JanLert, Lars-Erik, "Modelling Change—the Frame Problem." To appear in *The Frame Problem and Other Problems of Holism in Artificial Intelligence*, Zenon Pylyshyn, ed., Ablex Publishing, 1985.

[Kibler] Kibler, Dennis, and Morris, Paul, "Don't be Stupid." *IJCAI-81.*

[London-planner] London, Phil, "A Dependency-Based Modelling Mechanism for Problem Solving." Computer Science Technical Report 589, University of Maryland, College Park, Maryland, November 1977.

[London-representation] London, Phil, "Dependency Networks as a Representation for Modelling in General Problem Solvers." Computer Science Technical Report 698, University of Maryland, College Park, Maryland, September 1978.

[McDermott-time] McDermott, Drew. "Generalizing Problem Reduction: A Logical Analysis." *IJCAI-83*. .

[Milne] Milne, A.A., *Winnie The Pooh.* Dell Publishing Company, New York, 1984. First copyright 1926.

[GPS] Newell, A., Shaw, J.C., and Simon, H.A., "Report on a general problem-solving program." *Procedings of the International Conference on Information Processing*, pp. 256–264, UNESCO, Paris 1960. Reprinted in *Computers and Automation*, July 1959.

[Raphael] Raphael, Bertram, "The Frame Problem in Problem-Solving Systems." *Proceedings of the Advanced Study Institute on Artificial Intelligence and Heuristic Programming*, Menaggio, Italy, 1970.

[Rich-TR] Rich, Charles, *Inspection Methods in Programming.* MIT AI Technical Report 604, Cambridge Mass., June, 1981.

[Rich-representation] Rich, Charles, "A Formal Representation for Plans in the Programmer's Apprentice." *IJCAI-81*, pp. 1044–1052.

[Rosenschein] Rosenschein, Stanley J., "Plan Synthesis: A Logical Perspective." *IJCAI-81*, pp. 331–337.

[Rosenschein-multi-agent] Rosenschein, Jeffrey S., "Synchronization of Multi-Agent Plans." *AAAI-82*, pp. 115–119.

[NOAH] Sacerdoti, Earl D., *A Structure for Plans and Behavior.* SRI AI Technical Note 109, August, 1975. Also American Elsevier.

[Sacerdoti-plans] Sacerdoti, Earl D., "The Nonlinear Nature of Plans." *4th IJCAI*, pp. 206–214.

[Sacerdoti-tactics] Sacerdoti, Earl D., "Problem Solving Tactics." *nth IJCAI.* !!!

[Stallman-and-Sussman] Stallman, Richard M., and Sussman, Gerald Jay, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis." MIT AI Memo 380, September 1976.

[MOLGEN] Stefik, Mark Jeffrey, *Planning with Constraints.* PhD thesis, Stanford University, January 1980. Also Staford Heuristic Programming Project Memo 80-2 and Standford Computer Science Department Memo 80-784.

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

[CONNIVER] Sussman, Gerald Jay, and McDermott, Drew Vincent, "From PLANNER to CONNIVER—A genetic approach." *Proceedings of the Fall Joint Computer Conference*, 1972, pp. 1171-1179.

[HACKER] Sussman, Gerald Jay, *A Computational Model of Skill Acquisition.* MIT AI Technical Report 297, August 1973. Also American Elsevier.

[INTERPLAN-memo] Tate, Austin, "INTERPLAN: A plan generation system which can deal with interactions between goals." Machine Intelligence Research Unit Memorandum MIP-R-109, University of Edinburgh, Edinburgh, December 1974.

[INTERPLAN-IJCAI] Tate, Austin, "Interacting Goals and Their Use." *4th IJCAI.*

[INTERPLAN-thesis] Tate, Brian Austin, *Using Goal Structure to Direct Search in a Problem Solver.* PhD thesis, Univcrity of Edinburgh, 1975.

[NONLIN-IJCAI] Tate, Austin, "Generating Project Networks." *5th IJCAI*, 1977.

[NONLIN-TR] Tate, Austin, "Project Planning Using a Hierarchic Non-linear Planner." Department of Artificial Intelligence Research Report No. 25, University of Edinburgh, Edinburgh, August 1976.

[Tate-GOST] Tate, Austin, "Goal Structure—Capturing the Intent of Plans." *ECAI-84: Advances in Artificial Intelligence*, T. O'Shea, ed., Elsevier Science Publications B.V. (North-Holland), 1984.

[Tate-expert] Tate, Austin, "Planning in Expert Systems". Invited paper for the *Alvey IKBS Expert Systems Theme—First Workshcp* at Cosener's House, Abingdon, Oxford, 3-5 March, 1984. Also D.A.I. Research Paper 221, University of Edinburgh.

[DEVISER] Vere, Steven A., "Planning In Time: Windows and Durations for Activities and Goals." *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. PAMI-5, No. 3, May 1983. pp 246–267.

[Waldinger] Waldinger, Richard, "Achieving Several Goals Simultaneously." SRI Artificial Intelligence Center Technical Note 107, Menlo Park, July 1975.

[WARPLAN] Warren, David H. D., "WARPLAN: A System For Generating Plans." Department of Computational Logic Memo No. 76, Univerity of Edinburgh, Edinburgh, June 1974.

[Wilensky] Wilensky, Robert, "Meta-Planning: Representing and Using Knowlege About Planning in Problem Solving and Natural Language Understanding." *Cognitive Science* 5 (1981), pp. 197–233.

[SIPE-IJCAI] Wilkins, David E., "Representation in a Domain-Independent Planner." *IJCAI-83.*

[SIPE-AIJ] Wilkins, David E., "Domain-Independent Planning: Representation and Plan Generation." *Artificial Intelligence* 22:3 (1984) pp. 269–301. Also SRI International Technical Note No. 266R, Menlo Park, California, May 5, 1983.