From: AAAI-87 Proceedings. Copyright ©1987, AAAI (www.aaai.org). All rights reserved.

# Validating Generalized Plans in the Presence of Incomplete Information

Marianne Winslett*
Computer Science Dept., Stanford University
Stanford, CA 94305

**Abstract.** Let Robbie be an agent possessing a generalized plan for accomplishing a goal. Can Robbie use his plan to accomplish the goal without passing through any of a set of forbidden world states en route to the goal? This situation arises if, for example, Robbie must accomplish the goal with some additional constraints ("Can I get to the airport in time *without speeding*?").

There are two poles in the spectrum of methods Robbie can use to test his plan in the new world situation, each with its own advantages and disadvantages. At one extreme, Robbie can choose to express the new world constraints as additional preconditions on all the operators used for planning. At the other extreme, Robbie can attempt to prove that the new constraints are satisfied in every possible world that could arise during execution of the plan, from any initial world state that is consistent with his axioms. In this paper we examine the tradeoffs between these two opposing approaches, and show that the approaches are in fact very similar from a computational complexity point of view.

## 1. Introduction

Given a goal $\mathcal{G}$ that an agent will often need to achieve, it is natural to look for a means of reducing the time spent searching for a means to achieve $\mathcal{G}$. If the search space for $\mathcal{G}$ is large, then it may well be more efficient for the agent to store a *macro-operator* [Fikes 72] or a *skeletal* or *generalized* plan [Friedland 79, Schank 77, Stefik 80] for achieving $\mathcal{G}$, rather than searching through the problem space each time $\mathcal{G}$ and similar goals arise. We assume that this store-versus-compute controversy has been decided in favor of storage of a generalized plan for some of Robbie's goals, such as driving to the airport. Further, we assume that Robbie has a means of selecting a generalized plan relevant to the situation at hand and of binding the free variables in that plan to the appropriate entities for the current situation** [Alterman 86, Dean 85, Tenenberg 86]. The flow between operations in the resulting plan can be depicted graphically, as in the informal graph here of a simple plan for getting a drink of water.

A plan $\mathcal{P}$ is a sequence* of operators. A *path of execution* through $\mathcal{P}$ is a sequence of complete-information world states $S_0, S_1, \ldots, S_n$, where $S_0$ is the initial state of the world, where $S_j$ is obtained from $S_{j-1}$ by applying the $j$th operator in $\mathcal{P}$ to $S_j$, and where each $S_j$ is consistent with the agent's knowledge base (KB) and with a first-order encoding of $\mathcal{P}$ (described below).

Suppose that Robbie now must test whether his plan $\mathcal{P}$ for accomplishing goal $\mathcal{G}$ is still valid when there are new constraints on the permissible state of the world at each step of the execution of the plan. We assume that the new constraints can be formulated as a first-order formula $\alpha$, quantified over situations.** In Robbie's KB, it may well be the case that complete preconditions for successful execution of $\mathcal{P}$ have already been regressed [Waldinger 77] through all operators in $\mathcal{P}$, to form one initial overall precondition $\mathcal{C}$. In the airport example, $\mathcal{C}$ might dictate that Robbie have a driver's license and have easy access to a working automobile. Such a regression guarantees that the agent's goal (e.g., a timely airport arrival) can be attained from any initial world state that is consistent with the KB and with the instantiated form of $\mathcal{C}$. Unfortunately, this guarantee of correctness does not persist when the new constraint $\alpha$ on speeding is added to Robbie's KB, because even if the initial state of the world satisfies $\alpha$, some state Robbie goes through on a path to $\mathcal{G}$ might violate $\alpha$. Robbie might have a general plan to get to the airport such that getting there in time would unfortunately require speeding.

To see how this problem manifests itself in a for-

* ATT Doctoral Scholar; additional support was provided by DARPA under contract N39-84-C-0211.

** Of course, this is a research problem in its own right.

* This directly extends to plans with conditional application of operators and with operators with multiple possible outcomes.

** To simplify the presentation, we will make the restriction that the new constraint contain only one situation variable. For example, we will not consider constraints on transformations between situations.

malization of Robbie's plan, let us introduce some terminology that will be used throughout the remainder of this paper. We have already described $\mathcal{P}$ and $\mathcal{C}$, forms of a generalized plan and its overall precondition. The new constraint is denoted by $\alpha$. Situation variables and constants are written $s$, $s'$, $s_0$, $s_1$, etc. The situation describing the initial state of the world is $s_0$; Note that $s_0$ need not completely specify the state of the world; $S_0$ is a complete-information world state consistent with all knowledge about $s_0$. The result of applying the first operator in $\mathcal{P}$ to $s_0$ is situation $s_1$ (e.g., $s_1 =$ result(startUpCar($s_0$))); and so on until situation $s_n$, the final result of the plan, is defined.* The definitions of $s_0$ through $s_n$ constitute an *encoding* of $\mathcal{P}$. For any world state $S_i$ on a path of execution through $\mathcal{P}$, the situation corresponding to $S_i$ is situation $s_i$. Finally, for any formula $\phi$ with a single quantified situation variable $x$, let $\phi[s]$ be the formula created by binding $x$ to the constant $s$.

We now give a definition of plan validity. Assume that $\mathcal{C}[s_0]$ is true (e.g., ¬speeding($s_0$)). For $\mathcal{P}$ to be valid given the new constraint $\alpha$, $\alpha$ must be satisfied at *every* world state along *every* path of execution through $\mathcal{P}$. More formally, $\mathcal{P}$ is *valid* in the presence of the new constraint $\alpha$ if $\mathcal{C}[s_0]$ is true, and for every world state $S_i$ along every path of execution through $\mathcal{P}$, $\alpha[s_i]$ is true. Of course, one cannot prove validity by testing each possible $S_i$ separately, because the combinatorial possibilities are computationally overwhelming.

To check the validity of $\mathcal{P}$ more efficiently, it might seem to suffice to check whether $\mathcal{G}$ is true in situation $s_n$—for example, to check whether Robbie arrived at the airport in time. Unfortunately, this is insufficient; as long as $\mathcal{C}[s_0]$ is true, $\mathcal{G}[s_n]$ will be also.

For example, suppose that Robbie has a plan to conquer his thirst during dinner by getting a drink of water, as shown above. This can be represented in a simplified portion of Robbie's KB by using four predicates to describe the state of the world: atBldg(*bldg, sitn*), inRoom(*room, sitn*), thirsty(*sitn*), has(*item, sitn*); three plan operators: goToKitchen, pour, drink; and frame axioms telling what aspects of world state are not affected by application of operators. The resulting KB fragment appears in Figure 1.**

* A similar coding using conditionals can be used for plans with non-sequential structure.

** This is certainly not intended as a definitive or prescriptive encoding of thirst-quenchery; rather, it is a simple encoding that is sufficient for our purposes. We have omitted some important rules, such as type information, have lazily represented rooms within buildings as constants rather than functions, and have simplified the "pour" and "drink" operators.

Initial situation:

thirsty($s_0$)

inRoom(diningRoom, $s_0$)

Operators:

$\forall s$ inRoom(kitchen, result(goToKitchen, $s$))

$\forall s$ [inRoom(kitchen, $s$) $\rightarrow$ has(water, result(pour, $s$))]

$\forall s$ [has(water, $s$) $\rightarrow$ ($\neg$ thirsty(result(drink($s$))) $\wedge$ $\neg$has(water, result(drink($s$))))]

Frame axioms:

$\forall s$ [thirsty(result(goToKitchen, $s$)) $\leftrightarrow$ thirsty($s$)]

$\forall s$ $\forall x$ [has($x$, result(goToKitchen, $s$)) $\leftrightarrow$ has($x$, $s$)]

$\forall s$ $\forall x$ [atBldg($x$, result(goToKitchen, $s$)) $\leftrightarrow$ atBldg($x$, $s$)]

$\forall s$ [thirsty(result(pour, $s$)) $\leftrightarrow$ thirsty($s$)]

$\forall s$ $\forall x$ [$x \neq$ water $\rightarrow$ (has($x$, result(pour, $s$)) $\leftrightarrow$ has($x$, $s$))]

$\forall s$ $\forall x$ [$x \neq$ water $\rightarrow$ (has($x$, result(drink, $s$)) $\leftrightarrow$ has($x$, $s$))]

(other frame axioms showing that *atBldg* and *inRoom* are unaffected by pouring and drinking)

Additional axioms:

$\forall s$ $\forall x$ $\forall y$ [(atBldg($x$, $s$) $\wedge$ atBldg($y$, $s$)) $\rightarrow$ $x = y$]

$\forall s$ $\forall x$ $\forall y$ [(inRoom($x$, $s$) $\wedge$ inRoom($y$, $s$)) $\rightarrow$ $x = y$]

*Figure 1. Simplified portion of agent's KB.*

Then Rosie asks Robbie if he knows how to get water during dinner with the additional constraint that in a restaurant, Robbie should never be in the kitchen. The new constraint and the resulting encoding of plan $\mathcal{P}$ are shown in Figure 2.

Encoding of plan $\mathcal{P}$:

$s_1$ = result(goToKitchen, $s_0$)

$s_2$ = result(pour, $s_1$)

$s_3$ = result(drink, $s_2$)

New constraint $\alpha$:

$\forall s$[atBldg(restrant, $s$) $\rightarrow$ $\neg$inRoom(kitchen, $s$)]

Instantiated goal $\mathcal{G}[s_n]$:

$\neg$thirsty($s_3$)

*Figure 2. Plan to quench thirst, and a new constraint.*

Is $\mathcal{P}$ still valid no matter whether Robbie is at home or at a restaurant? Obviously not, because there is one path of execution through $\mathcal{P}$ in which Robbie is in a restaurant kitchen. Let $KB^+$ be Robbie's KB plus $\alpha$ and the encoding of $\mathcal{P}$ in Figure 2. Then invalidity of $\mathcal{P}$ cannot be detected by a test for logical consistency

of $KB^+$, because $KB^+$ is provably logically consistent.* Further, $KB^+$ logically implies $\neg$thirsty($s_3$), so one cannot detect invalidity by a test for provability of $\mathcal{G}$.** We conclude that, in general, simple checks for consistency are insufficient to show validity when new constraints are introduced.

In restricted cases, however, a simple check for consistency does suffice. If $\mathcal{P}$ is completely invalid, in the sense that *every* path of execution through $\mathcal{P}$ passes through a world state that is inconsistent with $\alpha$, then $KB^+$ will be inconsistent. For example, suppose the KB contains the additional formula atBldg(restaurant, $s_0$). From $KB^+$ one can prove, as before, $\neg$atBldg(restaurant, $s_0$); hence $KB^+$ is inconsistent. This implies that $\mathcal{P}$ will be valid if (1) the initial state of the world is completely determined by the KB, and $\mathcal{C}[s_0]$ is true; (2) all operators in $\mathcal{P}$ are deterministic; and (3) $KB^+$ is consistent. Unfortunately, as shown by Robbie's thirst-quenching plan, this approach does not extend to the common case where unknown, missing, or incomplete information is relevant to $\mathcal{P}$.

## 2. Prove-Ahead and Prove-As-You-Go

Assuming that an agent wishes to validate a plan completely before beginning its execution, there are two main approaches to an efficient and general means of plan validation. The first is to regress constraint $\alpha$ through all KB operators to form additional preconditions on those operators. In other words, add additional preconditions to each operator $\mathcal{O}$ so that $\mathcal{O}$ can never be applied if the resulting situation would violate $\alpha$. In the restaurant example, this can be done by adding an additional condition on the goToKitchen operator, so that Robbie cannot go into the kitchen if he is in a restaurant. We call this the *prove-ahead* approach, because we find the possible effects of $\mathcal{O}$ on $\alpha$ ahead of time and act to prevent violations of $\alpha$. After this regression phase is complete, we can test the plan for validity by either of two methods: either regress to a new overall plan precondition $\mathcal{C}'$ and check provability of $\mathcal{C}'[s_0]$, or else step through the plan operations and test whether the new preconditions of those operators are satisfied at each stage of execution.

There is a philosophical motivation for the pure prove-ahead approach, in which all constraints are regressed through all KB operators: once a complete regression has been done, any plan where $\mathcal{G}[s_n]$ is provable is a valid plan, no matter what the initial world state and no matter what branching occurs during plan execution. With pure prove-ahead, Robbie need not worry about detecting constraint violation at plan generation and validation time; he need only check preconditions.

The alternative to the prove-ahead approach in this example is to prove that at each situation on a path of execution through $\mathcal{P}$, Robbie is not in a restaurant kitchen. More formally, given that axiom $\alpha$ is satisfied in an initial situation $s_0$, one must prove that $\alpha$ is also true in situation $s_1$. If one can prove $\alpha[s_i]$ for each situation $s_i$ in the encoding of $\mathcal{P}$, then $\mathcal{P}$ is valid. We call this technique the *prove-as-you-go* approach, because we step through the operators of the plan in order, and for each operator $\mathcal{O}$ prove that $\alpha$ is true in the situation that results from applying $\mathcal{O}$ to the previous situation.

The remainder of this paper is a discussion of the advantages and disadvantages of the prove-ahead and prove-as-you-go approaches. We show that these two paradigms are at opposite ends of a spectrum of approaches, yet are computationally quite similar.

## 3. The Qualification Problem

Both the prove-ahead and prove-as-you-go approaches are methods of dealing with the *qualification problem* [Ginsberg 87, McCarthy 69, McCarthy 80]: what preconditions must be met in order for an action to succeed? In the real world, it is impossible to enumerate all the factors that might cause failure of a plan such as for a trip to the airport. This means that the philosophical motivation behind prove-ahead must ultimately be frustrated: except in simple systems, one cannot enumerate all the prerequisites for an operator. In attempting to do so, one will simply clutter up the KB with a sea of inconsequential preconditions for operations, and lower the intelligibility of the KB for outside reviewers.

Even were an exhaustive list of preconditions available, one would not in general want to take the time needed to prove that all the preconditions were satisfied. This problem also arises in the prove-as-you-go approach, however; one may not be able to afford the expense of proving that all constraints will be satisfied after an operation is performed. In section 6, we will discuss the relative adaptability of prove-ahead and prove-as-you-go to partial testing of preconditions and constraints.

## 4. A Comparison of Computational Complexity

The computational complexity of prove-ahead and prove-as-you-go depends on Robbie's language and KB.

* A more reliable sign of trouble is that $KB^+$ logically implies that Robbie is not in a restaurant in state $s_0$. This is because the path of execution in which Robbie is initially in a restaurant gets pruned from the tree of possible plan executions, because it is inconsistent with $\alpha$.

** As mentioned earlier, if a KB logically entails $\mathcal{C}[s_0]$, then $KB^+$ must logically entail $\mathcal{G}[s_n]$.

Depending on the form of his KB, testing plan validity can be in any complexity class: from polynomial time worst case on up through undecidable. The goal of this section is not to differentiate between these classes, but rather to show that prove-ahead and prove-as-you-go are in the *same* complexity class for any given type of KB. We will do this by comparing the requirements that prove-ahead and prove-as-you-go impose on a theorem-prover. In specific KB and plan instances, prove-as-you-go may be less costly than prove-ahead; this is particularly true if the agent is not interested in repairing invalid plans. Because additional information about the state of the world may be available at the time an operator in the plan is applied, prove-ahead does not detect invalidity as quickly as does prove-as-you-go.*

This tardy detection of invalidity arises because pure prove-ahead requires two rounds of proofs. Robbie must first regress $\alpha$ into new preconditions, and then test whether the new preconditions are satisfied during execution of $\mathcal{P}$. While prove-ahead may make the same set of calls to a theorem-prover as does prove-as-you-go, prove-ahead will not discover that $\mathcal{P}$ is invalid until the second phase of its computation, when the new preconditions are checked against world state information. A hybrid approach can be used to overcome this flaw in part, but prove-ahead will still require two rounds of proofs.

Prove-ahead has another computational disadvantage when compared with prove-as-you-go, in that prove-as-you-go can take advantage of all the state information available when trying to prove satisfaction of $\alpha$. Prove-ahead, on the other hand, first derives a *most general* condition under which $\alpha$ will be satisfied, and then checks to see whether that condition holds in the situation at hand. For example, consider the constraint that the car stay on the road at all times while driving. General preconditions for this condition may be very difficult to find. On the other hand, it may be trivial to show that the car is on the road right now; for example, Robbie may have a primitive robotic function available that tells him that the car is now in the middle lane. To elucidate this point further, we describe a method of implementing prove-ahead and prove-as-you-go.

Let $s$ be a situation on a path of execution through $\mathcal{P}$, and let $s'$ be the situation resulting from applying the next operator $\mathcal{O}$ in $\mathcal{P}$ to situation $s$. (As usual, the state of the world in situations $s$ and $s'$ need not be fully determined by the KB.) The prove-as-you-go method requires that one prove $\alpha[s']$ given $\alpha[s]$. If the proof fails and $\mathcal{P}$ is not to be repaired, then the validation process terminates at this point. If the proof fails and $\mathcal{P}$ is to be repaired, then two repair tactics are possible: depending on the method used to establish $\alpha[s']$, the reason for the failure can be converted into additional preconditions on operators and/or additional conditions for $\mathcal{C}$. For example, in many cases verification of a universal constraint $\alpha$ can be reduced to testing a number of ground instantiations of $\alpha$ [Winslett 87]. If any of these ground instantiations is not provable, then this pinpoints a case in which $\alpha$ is violated. After this violation is repaired, the process is repeated, searching for another violation of $\alpha$ in situation $s'$.

The prove-ahead method also requires that one prove $\alpha[s']$ given $\alpha[s]$, with the additional proviso that $s$ can be any legal situation, not just one on a path of execution through $\mathcal{P}$. In other words, in attempting to prove $\alpha[s']$, one cannot use any state information about $s$ that could be deduced from $\mathcal{P}$. Further, failure to find a proof does not mean that $\mathcal{P}$ is invalid; invalidity can only be ascertained by repairing $\mathcal{O}$ and then checking its new preconditions against initial world state information. In addition, prove-ahead requires detection of *all* violations of $\alpha[s']$ before determining whether $\mathcal{P}$ is valid. For example, a prove-ahead approach to the restaurant constraint would generate the new precondition ¬inBuilding(restaurant, $s$) for the goToKitchen operator, even if the first step of Robbie's plan were to go home. Finally, once the regression is complete and any invalidities have been detected, repair of $\mathcal{P}$ is accomplished by adding additional constraints to $\mathcal{C}$.

In the worst case the potential computational advantages of prove-as-you-go will not materialize. For example, if there is no helpful state information available for situation $s$, then prove-as-you-go will not have that advantage over prove-ahead. More precisely, suppose $\mathcal{P}$ contains situations $s_0$ through $s_n$, and let $O_i$ be the $i$th operator of $\mathcal{P}$. Suppose the agent finds a prove-as-you-go proof of $\alpha[s_i]$ such that that proof does not contain any $s_j$, for $0 \leq j \leq n$, other than $s_i$ and $s_{i-1}$. Then prove-ahead is as easy as prove-as-you-go for operator $O_i$, as essentially the same proof may be used for prove-ahead.

## 5. Hybrid Approaches

In general, neither the pure prove-ahead nor the pure prove-as-you-go approach will dominate in efficiency; a hybrid approach will be much more satisfactory. For example, there is no need to regress $\alpha$ through an operator until it is actually used in a plan. Then if $\alpha$ is a temporary constraint (such as a prohibition on speeding

* A more accurate cost comparison must consider the *amortized* cost of prove-ahead (section 5).

while Rosie is in the car), a complete regression will not be done.*

Regression through the operators in a particular plan is one point in the spectrum between pure prove-ahead and pure prove-as-you-go, a hybrid between the two approaches; such intermediate points abound. For example, Robbie might deliberately choose not to regress $\alpha$ through a particular operator $\mathcal{O}$ even though $\mathcal{O}$ appears in the plan at hand; this might be advisable if $\alpha$ was a temporary constraint and/or no plan repair was contemplated. He might choose prove-as-you-go for certain pairs of operators and constraints, and apply prove-ahead to the remainder. If Robbie can predict how often a particular prove-as-you-go proof would be repeated in the future, he can use measures of storage cost and other, less tangible factors (see section 6) to estimate the amortized cost of prove-ahead over all repetitions of that proof [Lenat 79]. The amortized cost of prove-ahead can then be used as a basis for choice between prove-as-you-go and prove-ahead.

Robbie need not apply the two phases of prove-ahead sequentially; the gap between regression and new precondition testing in prove-ahead can be narrowed by partial merging of the generation of new preconditions and their testing. If Robbie begins by regressing $\alpha$ through the first operator $\mathcal{O}$ of his plan, then he can immediately check to see whether the new preconditions for $\mathcal{O}$ are true in state $s_0$. If the preconditions are not true, then Robbie knows that $\mathcal{P}$ is not valid, and can proceed to repair or else search for another plan. For even more rapid detection of invalidity, Robbie can check each new precondition as it is generated. Prove-ahead will still require two rounds of proofs, however.

Robbie can even choose dynamically between prove-ahead and prove-as-you-go. For example, he can validate the first few steps of $\mathcal{P}$ using prove-ahead, and then decide, on the basis of the additional information available at that point, that prove-as-you-go is the best choice for the remainder of the validation sequence.

## 6. Comparative Extensibility of Prove-Ahead and Prove-As-You-Go

As described earlier, prove-as-you-go has a computational advantage over prove-ahead in its use of all state information available at a stage of plan execution. The counterpart of this prove-as-you-go advantage is the possibility in the prove-ahead world of choosing overly general preconditions. For example, suppose Robbie has a constraint on successful driving that the car must stay on the road at all times. The exact preconditions for maintaining this constraint at each moment are very complicated; they depend on how far Robbie is from the edge of the road, how fast he is going, etc. It would be easier to forgo exact preconditions and use a simpler subsuming condition, and run the risk of possibly rejecting a plan due to too-strict preconditions.

Another disadvantage of prove-ahead is that it will generate many operator preconditions, and the exact relation of those preconditions to the situational constraints will not necessarily be clear. This has repercussions for Robbie's ability to explain his decisions to an outside agent. Robbie will need some means of tagging preconditions and their associated constraints; this is a second-order concept. One may well argue that in a model of human car-starting, unlikely and inconsequential preconditions should not be stored with the operators that they impact. Rather, a human would call on its deductive facilities in the event that the car failed to start, to try and trace the origin of the failure to a combination of unchecked constraints.

This tagging consideration assumes greater importance if we abandon the assumption that Robbie never makes a move without consulting his theorem-prover. For example, Robbie may assign numerical measures of importance to preconditions and constraints, based on the likelihood of their being violated in the current situation and on the magnitude of the repercussions of their violation [Finger 86]. Then Robbie can choose which preconditions and constraints to check before performing an action, based on the computational resources available to him and the importance of the checks. However, in order to assign measures of importance to preconditions, again it is necessary to tag preconditions and their associated constraints.* Otherwise the consequences of failing to check a precondition will be unclear, for without an additional round of proofs, Robbie cannot easily tell which constraints may be violated if a particular precondition is ignored.

For example, Robbie may decide that it is not worthwhile to look for a potato in the tail pipe before turning the ignition key, for violation of this condition is unlikely, and ignoring it will not have a disastrous effect on the state of the world anyway. In contrast, Robbie might check this constraint after a fierce Idaho

* In this case, Robbie should employ a means of tagging operators and their unregressed constraints, so that he can easily tell which constraints are guaranteed to be satisfied due to preconditions and which constraints will require further proof.

* Alternatively, this information could be stored as first-order formulas in a special section of the KB. Efficient utilization of this section would again imply the use of special control knowledge.

rainstorm, or if there had been a recent rash of highly explosive tailpipe potatoes.

## 7. Conclusions

Let Robbie be an agent faced with the task of validating a plan $\mathcal{P}$ in the presence of a new constraint $\alpha$. If the initial state of the world is fully determined by information on hand and $\mathcal{P}$ is deterministic, then $\mathcal{P}$ is valid iff Robbie's knowledge base (KB) is consistent with $\alpha$ and an encoding of $\mathcal{P}$. However, another means of validation is needed when Robbie has insufficient information about the current state of the world.

We have identified two extreme approaches to this validation problem. In the pure *prove-ahead* approach, $\alpha$ is transformed into additional preconditions on Robbie's KB operators, and then the new operator preconditions for plan $\mathcal{P}$ are checked to see if they are true in the current situation, either by regressing those conditions or by direct checks. In the pure *prove-as-you-go* approach, to determine whether $\mathcal{P}$ is valid, Robbie must prove that $\alpha$ is true at each step of $\mathcal{P}$, given that $\alpha$ holds initially.

The prove-ahead and prove-as-you-go methods are computationally quite similar: for a given type of KB, they are in the same computational complexity class. In practice, prove-as-you-go may be less costly than prove-ahead, as its search for violations of $\alpha$ has a narrower focus. For example, any violation of $\alpha$ detected by prove-as-you-go may actually arise during the execution of $\mathcal{P}$; but prove-ahead may locate many potential violations of $\alpha$ that could not arise in $\mathcal{P}$ before finding those that do. This computational advantage arises because prove-as-you-go can utilize information about the state of the world that arises from prior operators in the plan. It may be much easier to prove that a constraint is satisfied at a particular point in the execution of a plan than to solve the prove-ahead problem of finding general preconditions for satisfaction of that constraint.

Pure prove-ahead and pure prove-as-you-go fall at two ends of a spectrum; in practice, we expect a hybrid approach to perform better than either extreme.

In choosing whether to apply prove-ahead or prove-as-you-go to a particular operator and constraint, one must consider factors other than simple computational complexity. Intuitively, the prove-as-you-go approach is best for "obscure" constraints. Prove-ahead is best for constraints that will be checked often, as under prove-as-you-go the same proofs would be performed time after time. An informed choice between prove-ahead and prove-as-you-go must consider the *amortized* cost of the two approaches, i.e., consider the number of times a proof would have to be repeated under prove-as-you-go, and also measure costs of storage, comprehensibility to outside agents, and extensibility to heuristic methods of planning: the store-versus-compute tradeoff once again.

## Acknowledgments

This paper arose from discussions of the planning problem with J. Finger, H. Hirsh, L. Steinberg, D. Subramanian, C. Tong, and R. Waldinger, who gave lively and patient arguments for the merits of and underlying motivations for the prove-ahead and prove-as-you-go approaches, and suggested directions for extensions.

## References

[Alterman 1986] R. Alterman, "An adaptive planner", *National Conference on Artificial Intelligence*, 1986.

[Dean 85] T. Dean, "Temporal reasoning involving counterfactuals and disjunctions", *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

[Fikes 72] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans", *Artificial Intelligence* **3**:4, 1972.

[Finger 86] J. Finger, "Planning and execution with incomplete knowledge", unpublished manuscript, 1986.

[Friedland 79] P. E. Friedland, *Knowledge-based experiment design in molecular genetics*, PhD thesis, Stanford University, 1979.

[Georgeff 85] M. P. Georgeff, A. L. Lansky, and P. Bessiere, "A procedural logic", *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

[Ginsberg 87] M. Ginsberg and D. E. Smith, "Reasoning about action I", submitted for publication.

[Lenat 79] D. B. Lenat, F. Hayes-Roth, and P. Klahr, "Cognitive economy in artificial intelligence systems", *Proceedings of the International Joint Conference on AI*, 1979.

[McCarthy 69] J. McCarthy and P. J. Hayes, "Some philosophical problems in artificial intelligence", in B. Meltzer and D. Michie, eds., *Machine Intelligence 4*, Edinburgh University Press, Edinburgh, 1969.

[McCarthy 80] J. McCarthy, "Circumscription—A form of non-monotonic reasoning", *Artificial Intelligence* **13**, 1980.

[Sacerdoti 77] E. D. Sacerdoti, *A structure for plans and behavior*, Elsevier North Holland, New York, 1977.

[Schank 77] R. C. Schank and R. P. Abelson, *Scripts, plans, goals, and understanding*, Lawrence Erlbaum, Hillsdale NJ, 1977.

[Stefik 80] M. J. Stefik, *Planning with constraints*, PhD thesis, Stanford University, 1980.

[Tenenberg 86] J. Tenenberg, "Planning with abstraction", *National Conference on Artificial Intelligence*, 1986.

[Waldinger 77] R. J. Waldinger, "Achieving several goals simultaneously", in E. W. Elcock and D. Michie, eds., *Machine Intelligence 8*, Halstead/Wiley, New York, 1977.

[Winslett 87] M. Winslett, *Updating databases with incomplete information*, PhD thesis, Stanford University, 1987.