Admissible Criteria For Loop Control In Planning

Roy Feldman and **Paul Morris**
IntelliCorp
1975 El Camino Real West
Mountain View, CA 94040
feldman@intellicorp.com morris@intellicorp.com

Abstract

We introduce methods for identifying operator preconditions that need not be expanded further. The methods are proved to be *admissible* , that is, they will not cause a solution to be missed when one exists. In certain cases, the methods also identify operator reformulations that increase the number of nonexpandable preconditions. This approach provides effective loop control in common situations. Moreover, the computation required can be performed during a precompilation of the operators in a domain; thus, there is no significant additional run-time overhead during planning.

Introduction

A major challenge facing the builder of a generative planner is to prevent undesired looping behavior. This is most clearly seen in "vicious circle" situations. For example, a planner, given the problem of opening a car door when the keys are locked in the car, may loop indefinitely in trying to find a solution. A more subtle form of the syndrome can occur during backtracking search. In these situations, time is wasted in considering partial solutions that involve unnecessary digressions. For example, a planner may consider going from one place to another by irrelevant circuitous routes.

Missing a solution due to infinite looping can be avoided by a breadth-first search, such as used by Tweak [Chapman 1987]. However, this kind of search is impractical for many types of problems, and in any event the efficiency issue remains.

Clearly, a planner needs some way of determining when a subgoal should not be expanded further. The danger of any pruning criterion, however, is that for some problems it may prevent any solution from being found, even when a solution exists. What is needed is a guarantee that the criterion allows at least one solution to be found for every solvable problem. A pruning strategy with this property is called *admissible* by Drummond and Currie [Drummond & Currie 1989].

Similar difficulties with looping can arise in backward-chaining inference systems. Smith, et al. [Smith, Genesereth & Ginsberg 1986] show that the common idea of failing a subgoal that is identical to one of its ancestors is an admissible strategy for inference systems. A similar approach has been used for loop control in planning (e.g., [Rich 1983, p. 258]). Unfortunately, as shown below, this strategy is not admissible for planning systems. Nonlinear planning can be formulated as an inference task via Chapman's modal truth criterion [Chapman 1987]. However, failing repetitive subgoals in the inferential problem appears to have little practical significance for the planning problem.

An alternative approach to loop control in planning was introduced by Tate [Tate 1976; Tate 1977]. In his NONLIN system, certain preconditions of operators are designated as *hold* conditions (called *usewhen* conditions in [Tate 1977]). Hold conditions are not allowed to be expanded. The difficulty here is that the user is required to specify these, and no guidance is provided for selecting them. If the user's intuition is faulty, the strategy will be inadmissible. Furthermore, as we will see presently, for some ways of specifying the operators in a domain, there may not be any admissible hold conditions that provide adequate loop control.

Nevertheless, the concept of a nonexpandable precondition will form the basis of our approach. We will present a way of determining when a precondition may admissibly be made nonexpandable. Moreover, the method automatically reformulates the operators in certain cases in order to increase the number of such determinations.

In following sections, we first show the inadmissibility in planning of failing subgoals that are identical to an ancestor. Then we present a formalization of plan graphs that serves as a framework for our approach. Next we introduce a structure called the *predecessor graph* that provides information about allowable operator sequences of length 2. Finally, we show how to use

This research was sponsored by a joint project of the Defense Advanced Research Projects Agency and the National Aeronautics and Space Administration under contract F30602-88-C-0045.

this graph to identify admissibly nonexpandable preconditions. The proofs are contained in the appendix.

Robot Recharging Problem

In this section, we present an example that resists existing methods of loop control. In particular, it shows the inadmissibility of failing a goal that is identical to an ancestor. The operator representation we will give for the problem, although a natural one, is also not amenable to control with hold conditions. We will see later that successful loop control requires a reformulation of one of the operators.

The example, called the "robot recharging problem," was first described in [Morris 1984]. A robot is capable of holding a unit of charge. As long as it is charged, the robot may move around. We assume that moving uses a negligible amount of energy. The robot also needs to fix a hole in its body. We assume this uses up all its energy, so that the robot becomes discharged. The robot is initially at location A. There is a power source at location B where the robot can recharge. The goals of the robot are to fix the hole and be charged.

The following is the STRIPS representation of the domain.

FIX
Preconditions and delete conditions:
CHARGED, HOLE
Add conditions: UNCHARGED, NO-HOLE
RECHARGE
Preconditions: AT(B), UNCHARGED
Delete condition: UNCHARGED
Add condition: CHARGED
GO
Preconditions: CHARGED, AT(?X)
Delete condition: AT(?X)
Add condition: AT(?Y)

Initial state: AT(A), CHARGED, HOLE
Goals: CHARGED, NO-HOLE

Obviously, the robot must first go to location B. Only then can it perform its task and still be able to recharge. However, if we examine the structure of this plan, we see that AT(B) is a subgoal of CHARGED. When AT(B) is expanded by a GO action, this introduces a further subgoal of CHARGED, which is identical to the ancestor goal. Note that if we fail this subgoal, the problem will not be solved. (It may be helpful to refer to figure 1 below.)

If we use no loop control method, the problem can be solved. Consider, however, the closely related problem where the robot is uncharged in the initial state. This has no solution. Without some loop control criterion, a planner faced with this problem will loop endlessly. The challenge is to find some method that cuts off the search in the second situation, but not in the first.

The most obvious candidates for hold conditions are the UNCHARGED precondition in RECHARGE, the AT(?X) in GO, and the HOLE in FIX. Unfortunately, these are not sufficient to terminate looping in the second situation, since there is a cycle through the AT(B) precondition of RECHARGE and the CHARGED precondition of GO. Intuitively, it seems as if the AT(B) in RECHARGE might be a hold condition. However, we saw above that constructing the plan in the first situation requires AT(B) to be expanded.

Notice that the need for the GO action indirectly results from the presence of the NO-HOLE goal, but this is not reflected in the goal ancestor path. Interestingly, the methods we will consider have the effect of reformulating the operators in a way that diverts the ancestor path to this goal.

Plan Graphs

We use the general framework of nonlinear planning introduced by Sacerdoti [Sacerdoti 1977] and Tate [Tate 1977]. Planning proceeds in stages. At each stage, a goal is selected to work on. The planner satisfies the goal either by matching it to an already achieved fact, or by introducing a new action into the plan. In the former case, we say the goal is *linked* to the matching fact. In the latter case, we say the goal is *expanded*.

The possible actions are determined by a set of STRIPS operators [Nilsson 1980]. However, we require that the operators be formulated so that they are never applied in situations where one of their delete conditions is already false, or where one of their add conditions is already true; i.e., none of the adds or deletes are no-ops. This is not a severe restriction. First, operators that are constructed in practice generally have this property. Second, an operator that does not can always be replaced by several more specialized operators that do satisfy the restriction. We also require that the operators be formulated to avoid the need for *coincident* solutions, that is, where two preconditions, or two add conditions, of the same operator become instantiated in such a way that they coincide. Again, this can be accomplished by using more specialized operators, if necessary.

We associate each action with two sets of facts: the *before-facts*, consisting of the delete conditions and undeleted preconditions; and the *after-facts*, comprised of the add conditions and undeleted preconditions. This division may be viewed as a relational production [Vere 1977] representation of the action.

A *plan graph* is a graphical representation of a plan that shows how the after-facts of various actions contribute to the before-facts of other actions. Formally, a plan graph is a directed acyclic multigraph where the nodes are labeled with actions, and the edges are labeled with facts. The labeling on an edge connecting two actions must belong to the after-facts of the first action and the before-facts of the second. There is also a distinguished START action whose after-facts

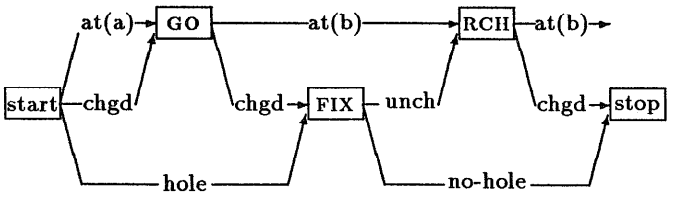

Figure 1: Plan Graph For Robot Problem

supply the initial facts for the plan, and a STOP action whose before-facts include the atomic top-level goals. Figure 1 shows the plan graph for a solution to the robot problem.

For linguistic convenience, we blur the distinction between a node and the action it is labeled with. For example, we may speak of the "before-facts" of a node, and the "incoming edges" of an action.

The following concepts will be useful in the subsequent discussion.

Note that every directed acyclic multigraph determines a partial order on its nodes and edges. Given a plan graph, we say a node or edge *precedes* another node or edge if it precedes it in this partial order. Furthermore, we say a node or edge is *parallel* to a second node or edge if neither precedes the other.

A plan graph is *complete* if the before-facts of each action in the plan are supplied by after-facts of other actions, i.e., if the incoming edges to each action cover all of the before-facts. This corresponds to a situation where there are no remaining unsolved goals. Unless otherwise stated, in the remainder of this paper, we will assume plan graphs are complete. Also, unless otherwise clear from the context, we restrict our attention to plan graphs that are *conflict-free* in the following sense: no edge is parallel to an action that deletes the fact that the edge is labeled with. This is closely related to the usual concept of plan conflict (e.g., [Tate 1976]).

It is never necessary for two incoming edges to an action to be labeled with the same fact, since we can simply drop one of the edges without affecting the completeness of the plan. Expressed in terms of planning, a goal is satisfied by linking to a single matching fact. However, traditional approaches to planning allow a fact to satisfy more than one goal. Thus, it is possible for two outgoing edges from an action to be labeled with the same fact. We will call this a *collision*. The following result indicates that collisions are not essential, and could be excluded without seriously impairing a planner.

Theorem 1 *For every conflict-free plan graph, there is a collision-free plan graph with the same actions that solves the same problem.*

In light of theorem 1, *we assume from now on that the plan graphs are collision-free.* This restriction turns out to be very useful for proving results about plan graphs.

Note that the theorem shows it is admissible to adopt an approach to planning where a fact that is linked to a goal is "used up" by the goal, i.e., becomes unavailable for linking by another goal. In the case where the consuming goal corresponds to a precondition of an action that remains true after the action, the fact is "put back" as a postcondition of the action. This approach to planning has been studied in [Morris 1984].

We now introduce a notion that is related to the concept of primary cut introduced by Drummond and Currie [Drummond & Currie 1989]. We will say a set of edges is *parallel* if every edge in the set is parallel to every other edge in the set. A *cut* of a plan graph is a maximal parallel set of edges, i.e., a parallel set of edges that is not a subset of any other parallel set of edges.

The following theorem (actually, its corollary) is central to reasoning about plan graphs.

Theorem 2 *For every cut, there is a reachable state of the domain that contains each fact in the labeling of the cut. Moreover, no two edges in the cut are labeled with the same fact.*

Corollary 2.1 *Edges that are parallel cannot have the same label or contradictory labels. A node cannot be parallel to an edge whose label coincides with or conflicts with one of its before-facts or after-facts.*

To apply the corollary, we need to reason about inconsistencies. In principle, any domain constraints could be used for this purpose. However, we have found the most useful constraints to be exclusive-or relationships. Inconsistencies based on such constraints can be efficiently determined. In [Morris & Feldman 1989], a method is presented for automatically extracting candidate exclusive-or relationships from operator descriptions.

The Predecessor Graph

Obviously, not every combination of actions is possible or useful. We now address the question of what actions can appear next to each other in a plan graph.

Given a plan graph, we will say a node A is an *immediate predecessor* of a node B if A precedes B, and there is no other node C such that A precedes C and C precedes B. We are interested in necessary conditions for a node to be an immediate predecessor.

It is easy to see that if A is an immediate predecessor of B, then some outgoing edge of A must be an incoming edge of B. (The converse is not necessarily true: A could be connected by an edge to B without being an immediate predecessor, since there may be a second route from A to B.) Corollary 2.1 provides additional requirements. Note that the after-facts of A must be

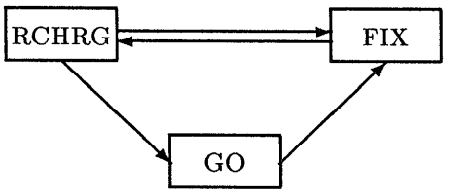

Figure 2: Predecessor Graph For Robot Problem

consistent with the before-facts of B; otherwise, we would have parallel contradictory edges.

This requirement allows us to completely rule out certain combinations of operators. For example in the robot recharging problem, no instance of GO can be an immediate predecessor of RECHARGE because the CHARGED precondition of the former conflicts with the UNCHARGED precondition of the latter. Similarly, FIX cannot immediately precede GO or itself, and RECHARGE cannot immediately precede itself.

Certain other combinations, although not impossible, can be ruled out as not being sensible. Suppose, for example, an action A is immediately preceded by its exact inverse B. Since the actions are inverses, there is an exact match between the after-facts of B and the before-facts of A. Corollary 2.1 then implies that every outgoing edge of A must be an incoming edge of B and vice versa; otherwise we would have parallel edges with the same label. It follows that the plan graph could be simplified by excising the portion from A to B and directly joining the resulting dangling edges.

There are no inverses in the robot problem. However, a further example of a non-sensible situation occurs when a two operator combination is subsumed by a single operator. Suppose in the robot problem that GO is immediately preceded by itself. The binding of ?Y in the earlier instance of GO must match the binding of ?X in the other; otherwise the consistency requirement is violated. But then the after-facts of the earlier instance coincide with the before-facts of the other, and all the edges from the former must connect to the latter. It follows that the portion of the plan graph containing the two GOs can be replaced by a single GO. Thus, we can assume that GO does not immediately precede itself.

After using the techniques above to eliminate various combinations, we are left with a set of combinations that have not been ruled out. We can form a graph of the operators that reflects these. We call this the *predecessor graph.* Figure 2 shows the predecessor graph for the robot problem.

The predecessor graph can often be annotated with constraints on bindings that arise from the immediate predecessor relationship. For example, if GO is immediately preceded by RECHARGE, then uniqueness of location requires that the variable ?X in GO be bound to B.

The above uses consistency analysis to rule out potential operator combinations in forming the predecessor graph. Other applications of consistency analysis in planning appear in the work of Irani and Cheng [Irani & Cheng 1987], and Drummond and Currie [Drummond & Currie 1989].

Loop Control

We now consider how to use the predecessor graph to determine nonexpandable preconditions. The method is based on results in this section.

We define a plan graph to be *minimal* if it contains no unnecessary actions, i.e., there is no other plan graph that solves the same problem with a proper subset of the actions. To prove admissibility of a pruning criterion, it is enough to show that every minimal plan graph can be constructed within the bounds of the criterion.

The following concept will also be useful. We say an operator A is *guarded* by one of its preconditions G if, for every minimal plan graph in which A occurs, the before-fact of A corresponding to G is an after-fact or a before-fact of every immediate predecessor of A.

Our first theorem on loop control involves a special case of A being guarded by G.

Theorem 3 *It is admissible not to expand a precondition G of an operator A if, for every minimal plan graph in which A occurs, the before-fact of A corresponding to G is a before-fact of every immediate predecessor of A.*

Intuitively, theorem 3 is used to make a operator precondition nonexpandable by way of deferral. If a precondition G meets the conditions of the theorem, it means that the expansion of any of the sibling preconditions of G are guaranteed to introduce a precondition identical to G. Thus, G can be expanded later. Our next result presents an alternative means of determining nonexpandability.

Theorem 4 *It is admissible not to expand a deleted precondition G of an operator A if A is guarded by G and, for every minimal plan graph in which A appears, the facts added by A are contained in the before-facts of every immediate predecessor of A.*

Moreover, this criterion and that of theorem 3 are simultaneously admissible.

The proof of theorem 4 is based on showing the expansion of the precondition is unnecessary because either some other goal can be expanded to introduce the same action, or any possible expansion will produce a non-minimal plan graph.

The following examples show how these theorems may be used. Our first example is a formalization of the car keys problem. We have the following operator descriptions.

OPEN-CAR-DOOR
Preconditions: DOOR-CLOSED, HAVE-KEYS
Delete condition: DOOR-CLOSED
Add condition: DOOR-OPEN
GET-KEYS-FROM-CAR
Preconditions: KEYS-IN-CAR, DOOR-OPEN
Delete condition: KEYS-IN-CAR
Add condition: HAVE-KEYS

In this example, consistency analysis shows that neither operator can have any immediate predecessor, i.e., all operator sequences of length 2 are impossible. Thus, every precondition can admissibly be made nonexpandable. This gives us the loop control we need.

Our next example is the robot problem. Consider the AT(?X) precondition of GO. The only immediate predecessor of GO is RECHARGE, which has AT(B) as a precondition. We noted earlier that when GO is immediately preceded by RECHARGE the ?X in GO must be bound to B. Thus, the precondition is also a precondition of every possible immediate predecessor. We conclude by theorem 3 that AT(?X) is admissibly nonexpandable for this domain.

Also, consider the UNCHARGED precondition of RECHARGE. The only immediate predecessor is FIX. Thus, UNCHARGED guards RECHARGED. Note that the only fact added by RECHARGE is CHARGED. This is among the preconditions of FIX. Moreover, UNCHARGED is deleted by RECHARGE. Thus, by theorem 4, it is a nonexpandable precondition.

We would like to make the AT(B) precondition of RECHARGE nonexpandable. However, as things stand the method does not apply, since the immediate predecessor, FIX, does not have AT(?X) as a precondition. Indeed, we noted earlier that with this operator representation, AT(B) needs to be expanded to solve the robot problem.

Observe, however, that if we include AT(?X) as an additional precondition in the FIX operator, then the criterion of the theorem is satisfied, and AT(B) can be made a nonexpandable precondition of RECHARGE. Moreover, it is admissible to augment the FIX operator in this way, since every state must satisfy some instance of AT(?X). The reason why the robot problem is now solvable is that the AT(?X) precondition of FIX can be expanded instead. Note that we have shifted the goal ancestor path for the expansion of AT(?X) so that it now leads to the NO-HOLE top-level goal. Intuitively, the problem with the original representation was that the expansions were not correctly motivated. Figure 3 shows the reformulated plan graph for this problem.

This technique of modifying the operators can be made systematic. In general, we may need to replace the immediate predecessor operator with several operators that are augmented with the different members of some exclusive-or set that includes the precondition under consideration. (If no better exclusive-or set is available, the fact and its negation can be used.) Note that consistency requirements ensure that of the replacement operators, only the one corresponding to the precondition is an immediate predecessor.

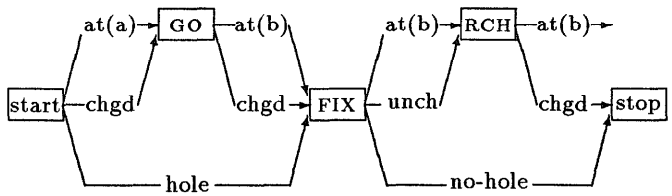

Figure 3: Reformulated Plan Graph

It is easy to see that the modification procedure can be applied to make any precondition nonexpandable provided that no immediate predecessor has the precondition as an add condition.

Observe that all the preconditions of RECHARGE are now nonexpandable. It can be verified that every potential loop in the robot problem passes through the RECHARGE operator. Thus, no loops remain.

It is important to note that the methods provide only sufficient conditions for admissible nonexpandability, and it is not true in general that they will prevent all loops. We can think of theorem 3 as eliminating goal repetitions where a goal is an immediate subgoal of itself. Theorem 4 applies to goals that repeat after two steps. Slightly larger repetition intervals can be handled by combining operators as necessary to reduce the repetition distance. However, in the blocks world, for example, there can be repetitions with an indefinite number of intervening goals, and neither theorem applies in this case. Nevertheless, the results here appear to cover many of the intuitive examples of hold conditions.

Closing Remarks

One of the features which distinguishes modern work in AI from that of earlier periods is an increasing concern for rigorous analysis and deeper understanding of the techniques studied. Previous loop control methods in planning may be classified as engineering art. Indeed, loop control is an important, but neglected issue in generative planning. We have presented provably correct methods that are effective in common situations. An important observation that follows from the results in this paper is that operator descriptions may need to be reformulated in order to facilitate control. The methods of the paper can be used to identify useful reformulations.

A system that identifies nonexpandable preconditions based on the theorems in the paper has been successfully implemented. Our present implementation uses a full theorem prover (the Boyer-Moore the-

orem prover [Boyer & Moore 1979]) to reason about inconsistency, which limits its applicability. However, in practice, less costly reasoning about exclusive-or relationships should be sufficient. Note also that the reasoning is performed during a precompilation of the operators in a domain, so the cost is amortized over all problem-solving in that domain.

Nonexpandability is only part of the content of the hold condition idea of Tate; the other part is a determination that linking need not be delayed until side effects of other operators are available. This property is called *effective isolation* in [Morris 1984]. The predecessor graph may also be useful for determinations of effective isolation, and current work is exploring this.

Acknowledgment We thank Bob Filman for suggestions to improve the presentation.

Appendix: Proofs

Theorem 1: For every conflict-free plan graph, there is a collision-free plan graph with the same actions that solves the same problem.

Proof: Suppose an action has two outgoing edges labeled with the same fact F. If the two edges are received by a single action, then one edge is redundant and can be removed. Otherwise, the edges must be received by different actions. At least one of these, say A, is not preceded by the other, B. If the plan is conflict-free, then the receiving action A cannot delete the fact F, since it is parallel to the edge that goes to B. Thus, F must also occur as an after-fact of A. Now the original F-labeled edge to B can be replaced by an edge from A to B. If there is already an outgoing edge from A labeled with F, then the above process can be repeated. Note that each repetition moves the collision closer to the STOP node. Ultimately, either it disappears, or a redundancy occurs among the incoming edges to the STOP node and can be removed.

Theorem 2: For every cut, there is a reachable state of the domain that contains each fact in the labeling of the cut. Moreover, no two edges in the cut are labeled with the same fact.

Proof: We say a node precedes a cut if it precedes some edge in the cut. The theorem will be proved by induction on the number of nodes that precede a cut.

Note that the outgoing edges from the START node constitute a cut that is preceded only by the START node itself. We call this the *initial cut.* Clearly, the result holds for the initial cut.

Now consider some other cut. Let A be a node preceding the cut that is maximally close to the cut, i.e., A does not precede any other node that precedes the cut. Let e be any outgoing edge of A. We show e must belong to the cut. Suppose otherwise. Then e cannot precede any edge in the cut; otherwise there would be another node between A and the cut. Also, e cannot be preceded by any edge in the cut. If it did, then A could not precede the cut. Thus, e is parallel to every edge in the cut. But this contradicts the definition of a cut, which requires it to be a maximal parallel set of edges.

We have shown that every outgoing edge of A belongs to the cut. Let us call this cut C. Now consider the set of edges obtained from C by replacing the outgoing edges of A with its incoming edges. It is not hard to verify that this is also a cut, which we will call C'. By the inductive hypothesis, the labels of edges in C' are free of duplicates, and are contained in some reachable state. The preconditions of A are satisfied in that state. Thus, the state resulting from applying A is also reachable. This state contains the facts corresponding to edges in C. This proves the first part of the result.

Note that the only way that applying A could produce a duplicate edge (we are assuming no collisions among the outgoing edges of A) would be if one of its outgoing edges had the same label as an edge common to the two cuts. But then A would be applicable in a situation where one of its add conditions is already true, contrary to our restriction on operators. This shows the second part of the result.

Corollary 2.1: Edges that are parallel cannot have the same label or contradictory labels. A node cannot be parallel to an edge whose label coincides with or conflicts with one of its before-facts or after-facts.

Proof: If two edges are parallel, then there is some cut that contains them both. By the theorem, the cut has no duplicate labels. Furthermore, the labels are contained in some reachable state, so they cannot be contradictory.

Now suppose a node A is parallel to an edge e. It is easy to see that the incoming and outgoing edges of A must also be parallel to e. Thus, the only case we need to consider is that of an unused after-fact. Note that the plan graph can be modified to include the after-fact as a top level goal without changing any of the labels. Then, the previous reasoning can be applied to the after-fact. The result follows.

Lemma 1 *Suppose A is guarded by a precondition G. Consider some minimal plan graph in which A occurs. Let e be the incoming edge to A that corresponds to G, and let B be the node of which e is an outgoing edge. Then B is an immediate predecessor of A, and is the only immediate predecessor of A in the plan graph.*

Proof: Clearly, B is a predecessor of A. Suppose there is some other node C that is an immediate predecessor of A. Then C has either an after-fact or a before-fact that matches the label of e. Clearly, e is parallel to C, leading to a violation of corollary 2.1. Thus, there cannot be any other immediate predecessor. It follows that B must be the immediate predecessor of A.

Theorem 3: It is admissible not to expand a precondition G of an operator A if, for every minimal plan graph in which A occurs, the before-fact of A corresponding to G is a before-fact of every immediate predecessor of A.

Proof: Consider a minimal plan graph in which A occurs. There must be some incoming edge e of A that corresponds to G. Suppose the edge e comes from another node B. By the statement of the theorem, A is guarded by G. Hence, by lemma 1, B must be an immediate predecessor of A.

By hypothesis, B has a before-fact that coincides with the label of e, and so e does not correspond to an add condition of B. It follows that during planning, B could not have been introduced into the plan as an expansion of G, i.e., it must have resulted from the expansion of some other goal (or be the START node). Thus, it is safe to make G nonexpandable.

Theorem 4: It is admissible not to expand a deleted precondition G of an operator A if A is guarded by G and, for every minimal plan graph in which A appears, the facts added by A are contained in the before-facts of every immediate predecessor of A.

Moreover, this criterion and that of theorem 3 are simultaneously admissible.

Proof: In the following, We use the term *add-fact* to describe an after-fact that corresponds to an add condition. Similarly, *delete-fact* is a before-fact that corresponds to a delete condition.

Consider any minimal plan graph in which a node B occurs. Let $\mathcal{E}$ be the set of outgoing edges of B that correspond to add-facts of B, and let $\mathcal{A}$ be the set of nodes that receive the edges in $\mathcal{E}$.

Each edge e in $\mathcal{E}$ corresponds to a precondition G of some node A in $\mathcal{A}$. We claim that at least one such G is not made nonexpandable by the criterion of the theorem. Assume otherwise. We will show that this leads to a contradiction. We note here that the assumption implies that every edge from an add-fact of B leads to a delete-fact of the connected node.

We proceed by showing all the outgoing edges of the nodes in the set $\mathcal{A} \cup \{B\}$ (other than internal edges) can be replaced by equivalent incoming edges. This will allow all the nodes in the set to be excised, contradicting the minimality of the plan graph.

By lemma 1, B is an immediate predecessor of each node in $\mathcal{A}$. It follows that the edges being considered for replacement are all parallel to each other. By corollary 2.1, their labels are all different. Thus, there is no danger of two outgoing edges competing for a replacement among the incoming edges, so it is sufficient to show that the edges can be replaced individually.

There are three cases to be considered: outgoing edges of (nodes in) $\mathcal{A}$ that correspond to add-facts; other outgoing edges of $\mathcal{A}$; and outgoing edges of B that do not go to $\mathcal{A}$.

If e corresponds to an add-fact of some A in $\mathcal{A}$, then by the condition of the theorem, it can be replaced by an equivalent incoming edge e' to B.

If e is an outgoing edge of some A, but does not correspond to an add-fact, then there must be an incoming edge e' of A that has the same label. If e' is not an internal edge, then e can simply be replaced by e'. Otherwise, e' is an outgoing edge of B. Since its label is not a delete-fact of A, it is not an add-fact of B (by the assumption; see above). Thus, there must be an incoming edge e'' of B with the same label as e'. In this case e can be replaced by e''.

If e is an outgoing edge of B that does not go to some A, then it is not in $\mathcal{E}$. It follows that it is not an add-fact of B. Thus, there is some incoming edge e' of B with the same label, and we can replace e with e'.

This proves our claim. Thus, at least one edge in $\mathcal{E}$ must correspond to a precondition G that does not satisfy the criterion of the theorem. When the plan has been constructed up to the after-facts of B, this G can be expanded to introduce B into the plan.

Note that the G exhibited here corresponds to an add-fact of B; thus, B does not have an equivalent before-fact. It follows that G is not made nonexpandable by the criterion of theorem 3.

References

Boyer, R. S., and Moore, J.S. *A Computational Logic.* Academic Press, 1979.

Chapman, D. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, July 1987.

Drummond, D., and Currie, K. Goal ordering in partially ordered plans. In *Proc. IJCAI-89*, pages 960–965, Detroit, Michigan, 1989.

Irani, Keki B., and Cheng J. Subgoal ordering and goal augmentation for heuristic problem solving. In *Proc. IJCAI-87*, pages 1018–1024, Milan, Italy, 1987.

Morris P., and Feldman, R. Automatically derived heuristics for planning search. In *Second Irish Conference on Artificial Intelligence and Cognitive Science*, Dublin, Ireland, 1989. Proceedings to appear in Springer-Verlag Brit. Comp. Soc. Workshop Series.

Morris, P. H. *A Resource Oriented Formalism for Plan Generation.* PhD thesis, University of California, Irvine, 1984.

Nilsson, N. J. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, Ca., 1980.

Rich, E. *Artificial Intelligence* McGraw-Hill, 1983.

Sacerdoti, E. D. *A Structure for Plans and Behavior.* Elsevier North-Holland, 1977.

Smith, D. E.; Genesereth, M. R.; and Ginsberg, M.L. Controlling recursive inference. *Artificial Intelligence*, 30(3):343–389, 1986.

Tate, A. *Project Planning Using A Hierarchic Non-Linear Planner.* Research Report 25, Dept. of Artificial Intelligence, Univ. of Edinburgh, Aug. 1976.

Tate, A. Generating project networks. In *Proc. IJCAI-77*, pages 888–893, Cambridge, Ma., 1977.

Vere S.A. Relational production systems. *Artificial Intelligence*, 8:47–68, 1977.