Formalizing Plan Justifications

Eugene Fink *
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L3G1
efink@violet.waterloo.edu

Qiang Yang *
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L3G1
qyang@logos.waterloo.edu

Abstract

This paper formalizes the notion of *justified plans*, which captures the intuition behind "good" plans. A justified plan is one that does not contain operators which are not necessary for achieving a goal. The importance of formalizing this notion is due to two reasons. First, it gives rise to methods for optimizing a given plan by removing "useless" operators. Second, several important concepts describing abstraction hierarchies are defined via justified plans. In the past, relatively few attempts have been made to formalize such a notion. This paper defines several different kinds of plan justifications, presents algorithms for finding a justified version of a plan, and shows that the task of finding the *best possible* justified version of a plan is NP-complete. Finally, it presents a greedy algorithm for finding a near-optimal justified plan in polynomial time.

1 Introduction

While searching for a plan that achieves a certain goal, we wish to find an efficient plan, which does not contain "useless" steps. Such a plan can be obtained from an inefficient plan by removing all operators that are not necessary for achieving the goal. For example, suppose that one wishes to prepare tea, by following the plan: "put a tea bag into a cup; boil water in a kettle; pour water into the cup". Suppose that later on one discovers that the kettle already contains hot water. Then the second step of the plan, "boil water", is no longer necessary for achieving the goal. After removing the second step, the resulting plan "put a tea bag into a cup; pour water into the cup" contains fewer steps while still achieving the same goal. The operation of removing useless operators from a plan is known as *justification*. The main purpose of our paper is to formalize different ways of performing plan justifications.

One application of plan justification is to augment a non-optimal planner such as STRIPS with an optimization routine. The resulting plan will then be more efficient to execute. Another application is reusing old plans. Suppose that we have found a plan for achieving goals G_1, G_2, and G_3. Later on we may use the same plan to achieve the goal G_1 alone. In this case we wish to find a subset of the initial plan which is "relevant" to achieving G_1, by removing all unnecessary operators. Thus, justification is useful for adapting an old plan to new situations.

The notion of justified plans is important not only for the purpose of optimizing plans, but also for abstract problem solving. Several important concepts describing the algorithms for generating abstraction hierarchies are defined via justified plans. For example, the theoretical concepts underlying Knoblock's planner ALPINE [Knoblock, 1990] are based on the notions of *justified plans*. Other results that depend on this notion are presented in [Tenenberg and Yang, 1990], [Knoblock *et al.*, 1991], and [Bacchus and Yang, 1991].

In spite of the importance of the concept of justified plans, relatively few efforts have been made to explore different kinds of justification. This paper begins to address this problem by formalizing and extending the previous work. We first consider the notion of *backward justified* plans that researchers have used before, which guarantees that each operator in a plan establishes a literal necessary for achieving a goal. We then present a definition of *well-justified* plans. Informally, a plan is well-justified if none of its operators may be omitted without violating the correctness of the plan. We also compare well-justified and backward justified plans in terms of their qualities. Finally, we consider the task of finding the "best possible" justification of a given plan, a subplan of a given plan that cannot be further optimized by removing any subset of its operators. We show that the task of finding such a subplan is NP-complete. To satisfy the practical need for efficient planning, we present a greedy algorithm that finds a near-optimal jus-

*The authors are supported in part by a scholarship and grants from the Natural Sciences and Engineering Research Council of Canada.

tification in polynomial time.

We begin by presenting a formal description of the problem space language used for describing our results. Then we consider each type of justification in turn.

2 Problem Space Language

A *planning domain* consists of a set of literals $\mathcal{L}$ and a set of operators O. Each *operator* α is defined by a set of *precondition literals* $Pre(\alpha)$ and *effect literals* $Eff(\alpha)$.

A *state* of the world is a set of literals. Applying an operator α to some state produces a new state, where all literals from $Eff(\alpha)$ hold, and all literals that do not conflict with $Eff(\alpha)$ are left unchanged. For example, suppose p_1 and p_2 are some atomic statements in a problem domain. The corresponding literals are p_1, p_2, $\neg p_1$, and $\neg p_2$. Let $Eff(\alpha) = \{\neg p_1\}$. Then applying α to the state $S = \{p_1, p_2\}$ produces the new state $S' = \{\neg p_1, p_2\}$.

A *linearly ordered plan* $\overline{\Pi} = (\alpha_1, \ldots, \alpha_n)$ is a sequence of operators, which can be applied to some *initial state* by executing each operator in order. A plan $\overline{\Pi} = (\alpha_1, \ldots, \alpha_n)$ is *legal* relative to an initial state S_0 if the preconditions of each operator are satisfied in the state in which the operator is applied, i.e. $\forall i \in [1 \ldots n]$, $Pre(\alpha_i) \subseteq S_{i-1}$. A plan $\overline{\Pi}$ *solves* a *goal state* S_g if $\overline{\Pi}$ is legal and the goal is satisfied in the final state: $S_g \subseteq S_n$. A legal plan that solves the goal S_g is called *correct* relative to S_g.

A *partially ordered plan* is a set of operators $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ with a partial order $\prec_\Pi$ on it. This partial order represents the time-precedence relation between operators: $\alpha_1 \prec_\Pi \alpha_2$ means that α_1 must be executed before α_2. A linearly ordered plan $\overline{\Pi}$ is a *linearization* of Π if it contains all the operators of Π and the order defined by $\prec_\Pi$ is not violated, that is for any α_i and α_j, if $\alpha_i \prec_\Pi \alpha_j$, then α_i occurs before α_j in $\overline{\Pi}$. A partially ordered plan is legal if all its linearizations are legal, and it solves a goal S_g if all its linearizations do. Throughout the remainder of the paper all plans are partially ordered unless otherwise specified.

A plan Π' is called a *subplan* of Π if it is obtained from Π by eliminating one or more operators. The precedence relation between the remaining operators must be preserved. That is, Π' is a subplan for Π if and only if

$\forall \alpha_1, \alpha_2 \in \Pi'$
(1) $\alpha_1, \alpha_2 \in \Pi$ and
(2) $\alpha_1 \prec_{\Pi'} \alpha_2 \Leftrightarrow \alpha_1 \prec_\Pi \alpha_2$.

3 Backward Justification

To formalize the notion of justified plans, we first generalize the concept of establishment defined in [Knoblock *et al.*, 1991] to partially ordered plans.

Definition 1 (Establishment) *Let $\overline{\Pi}$ be a legal linearly ordered plan. Let α_1 and α_2 be two operators of the plan $\overline{\Pi}$, $\alpha_1, \alpha_2 \in \overline{\Pi}$, $l \in \mathrm{Eff}(\alpha_1)$, and $l \in \mathrm{Pre}(\alpha_2)$. Then α_1* establishes l for α_2 *if*

1. $\alpha_1 \prec \alpha_2$, and
2. $\forall \alpha \in \overline{\Pi}$, if $\alpha_1 \prec \alpha \prec \alpha_2$ then $l, \neg l \notin \mathrm{Eff}(\alpha)$

We say that α_1 possibly establishes *a literal l for α_2 in a partially ordered plan Π if it establishes l for α_2 in at least one linearization of Π.*

Intuitively this means that the precondition l of the operator α_2 holds before the execution of α_2, and α_1 is the last operator that achieves it.

Definition 2 (Backward justification) *Let Π be a legal plan that achieves a goal S_g. An operator $\alpha \in \overline{\Pi}$ is called* backward justified *if $\exists l \in \mathrm{Eff}(\alpha)$ such that α possibly establishes l either for the goal S_g or for another backward justified operator.*

We say that a plan Π is backward justified, if all its operators are backward justified. This definition of justification was used in the planner ALPINE [Knoblock *et al.*, 1991]. For linearly ordered plans it is equivalent to the definition stated in [Tenenberg and Yang, 1990]. For partially ordered plans, backward justification is weaker then the justification described in [Tenenberg and Yang, 1990].

An operator is backward justified if it possibly establishes some literal necessary for achieving the goal. However, it may happen that l has already been established before α, and then α is useless in Π. Thus backward justified operators are not "truly justified". We illustrate this point with the following example.

Assume that one has a kettle with hot water and an empty cup, and wishes to have a cup of hot water. The following plan achieves the goal

1. Pour water into the cup.
2. Put the cup into a microwave.

The second operator is backward-justified, because it makes the water hot, while no other operator *after* it achieves the same goal. However, this operator may still be removed, because the water was already hot before its execution. Thus, the second operator is not truly justified.

Observe that if α is the last operator in some linearization of a plan that does not establish any goal literals or operator preconditions, then it can be removed without violating correctness of the plan. After its removal, the plan remains correct. We could then apply the same procedure recursively, until no more operators can be removed without violating the correctness of the plan. This is the basis of the algorithm for finding a backward justified plan.

The algorithm is shown in Table 3a. It first linearizes the plan Π. Then it checks whether or not the last operator α in the plan establishes a goal. If α doesn't establish any goal, then it should be removed. Then the algorithm considers the rest of the operators, going from the end to the beginning of the plan. Each operator that does not establish any literal for the goal nor for any other operator is removed. Observe that when we consider an operator, all operators after this operator that are not backward justified are already removed. Thus, the operator is not removed only if it establishes a literal for some *backward justified* operator, which means that the operator itself is backward justified. Since the algorithm proceeds from the end to the beginning of the plan, the resultant plan is called *backward* justified.

To check the condition in line 5, we need to check for every $\alpha_1 \in \Pi$ with the precondition l, if there is a linearization of the plan Π where no operator between α and α_1 establishes or removes the literal l. In other words, for each operator that achieves l or $\neg l$, we have to check whether it is necessarily between α and α_1[1]. If there is no such an operator, α possibly establishes l for α_1. If the order of operators is represented by a transitively closed graph, this condition may be checked in $O(|\Pi|)$ time for each α_1, where $|\Pi|$ is the number of operators in the initial plan. Therefore the search of α_1 established by α takes $O(|\Pi|^2)$ time. The overall running time of the algorithm is $O(E \cdot |\Pi|^2)$, where $E = \sum_{\alpha \in O} |\mathit{Eff}(\alpha)|$, and $|\mathit{Eff}(\alpha)|$ is the number of literals in the set of effects of α.

4 Well Justification

Definition 3 (Well-justification) *An operator α_i in a linearly ordered plan $\overline{\Pi}$ is called* well-justified *if $\exists l \in$* Eff(α_i) *such that α_i establishes l for some operator or for the goal S_g, and l does not hold before α_i, that is $l \notin S_{i-1}$.*

An operator in a partially ordered plan is called well-justified if it is well-justified in at least one linearization of the plan.

A plan is well-justified if all its operators are well-justified. Intuitively, an operator is well-justified if it establishes some literal which has not been established before, and which is necessary for executing some other operator. This means that if we remove a well-justified operator from a plan, the plan is no longer correct. We state this result as a lemma.

Lemma 1 *An operator is well-justified if and only if we cannot remove it from the plan without violating correctness of the plan.*

The next theorem follows directly from the lemma.

Theorem 1 *A plan is well-justified if and only if there is no operator that can be removed without violating correctness of the plan.*

This theorem shows that well-justification captures the intuition behind "good" plans: a well-justified plan does not contain any operator that is not necessary for achieving the goal. Recall that if a plan Π is not backward justified, then any operator that is not backward justified may be removed without violating the correctness of Π. By Theorem 1 this means that Π is not well-justified either. Thus, every well-justified plan is backward justified. In other words, well-justification is stronger than backward justification.

For a given legal plan, there might be several distinct well-justified subplans of the same plan, as the following example demonstrates.

Suppose one has a kettle of cold water, and needs a cup of hot water. The following plan would lead to the desired result

1. Boil water by putting the kettle onto a stove.
2. Pour the water into the cup.
3. Put the cup into a microwave.

This plan is not well-justified, because either the first or third operator may be skipped without violating the correctness of the plan. Thus, the plan has two well-justified subplans: one of them consists of the first two operators, and the other consists of the last two.

The simple algorithm that finds a well-justified subplan of a given plan is shown in Table 3b. The running time of the algorithm that checks correctness of a given plan is $O(P \cdot |\Pi|^2)$, where $P = \sum_{\alpha \in O} |\mathit{Pre}(\alpha)|$, and $|\mathit{Pre}(\alpha)|$ is the number of literals in the set of preconditions of α. Therefore the overall running time of the algorithm is $O(P \cdot |\Pi|^4)$.

5 Perfect Justification

While well-justified plans cannot contain unnecessary operators, they still may contain unnecessary *groups of operators*. This means that while no single operator may be eliminated from the plan, several operators may be eliminated *together*. In particular, a linearly ordered well-justified plan $\overline{\Pi} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ may contain a *cycle*, which means that the same state is achieved twice during the plan execution. Formally, a sequence of operators $\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_j$ in $\overline{\Pi}$ is called a cycle if $S_i \supseteq S_j$. Observe that we may eliminate a cycle from $\overline{\Pi}$ without violating correctness of $\overline{\Pi}$. For example, consider the following plan of boiling water:

1. Fill a cup with water.
2. Empty the cup.
3. Fill the cup with water again.
4. Put the cup into a microwave.

This plan is well-justified: we cannot skip *operator 2*, because then we could not fill the cup again; and we cannot skip *operator 3*, because the cup has to be full when we put it into a microwave. However, we may skip *operators 2* and *3* together. To formalize this observation, we introduce the notion of perfect justification.

Intuitively, a plan is perfectly justified if no subset of its operators may be removed from the plan. In other words, this is the "best possible" justification.

[1] An operator β is *necessarily* between α and α_1, if $\alpha \prec \beta$ and $\beta \prec \alpha_1$.

Definition 4 (Perfect justification) *A correct plan* Π *is called* perfectly justified *w.r.t. a goal* S_g *if it does not have any legal proper subplan that achieves the goal.*

Just by definition perfect justification is stronger than all justifications discussed above. Unfortunately, a perfect justification of a given plan cannot be found in polynomial time. In this paper we show that the task to find a perfect justification of a given plan is NP-hard, even for linearly ordered plans. Moreover, it is NP-hard to check whether a linearly ordered plan is perfectly justified.

Theorem 2 *Suppose we are given a linearly ordered plan* $\overline{\overline{\Pi}}$ *with an initial state* S_0 *and a goal* S_g*, and we wish to determine whether this plan is perfectly justified. This problem is NP-complete.*

Sketch of the proof. The problem is trivially NP, since, given a subplan of $\overline{\overline{\Pi}}$, we may check whether this subplan is legal and achieves the goal in polynomial time. To show that the problem is NP-hard, we reduce 3-clause satisfiability problem to our problem.

Suppose we are given a 3-clause conjunctive normal form with n distinct variables $V_1, V_2, \ldots, V_n$, and k distinct clauses $C_1, C_2, \ldots, C_k$. For each variable V_i we introduce two predicates, v_i^+ and v_i^-. For each clause C_j we introduce a corresponding predicate c_j. Finally, for each pair (V_i, C_j), where V_i is a variable *in* the clause C_j, we introduce a predicate p_{ij}. We define a problem domain that contains all literals defined by the introduced predicates. (Each predicate p gives rise to the two literals: p and $\neg p$.) We define operators in our problem domain as shown in Table 1.

We define an initial state S_0 as follows

∘ $(\forall i \in [1 \ldots n])\ v_i^- = \mathit{True}$ and $v_i^+ = \mathit{False}$

∘ $(\forall j \in [1 \ldots k])\ c_j = \mathit{False}$

∘ for all predicates p_{ij} in our domain, $p_{ij} = \mathit{True}$

and a goal S_g as follows:

∘ $(\forall j \in [1 \ldots k])\ c_j = \mathit{True}$

∘ for all predicates p_{ij} in our domain, $p_{ij} = \mathit{True}$

Now we present a linearly ordered plan with the initial state S_0:

$$\overline{\overline{\Pi}}=(\alpha_1, \alpha_2, \ldots, \alpha_n,\ \delta,\ \text{all } \beta_{ij}\text{'s, all } \gamma_{ij}\text{'s})$$

where the order of β_{ij}'s and γ_{ij}'s is arbitrary. It is straightforward to verify that the plan $\overline{\overline{\Pi}}$ is legal and solves the goal S_g. Further, one may show that *if* $\overline{\overline{\Pi}}$ has a legal proper subplan that achieves the goal, this subplan cannot contain δ, for if some of α-operators is removed from $\overline{\overline{\Pi}}$, the preconditions of δ are not satisfied, and if one or more β's or γ's are removed, δ interferes with achieving the goal. Thus, if Π has a legal proper subplan, this subplan must have the form

$$\overline{\overline{\Pi}}'=(\alpha_{k_1}, \alpha_{k_2}, \ldots, \alpha_{k_m},\ \text{some } \beta_{ij}\text{'s, some } \gamma_{ij}\text{'s})$$

It may be shown that the following two statements are equivalent:

- $\overline{\overline{\Pi}}$ has a legal subplan of the form $\overline{\overline{\Pi}}'$ that achieves the goal
- $V_{k_1} = V_{k_2} = \ldots = V_{k_m} = \mathit{True}$, and all other variables $V_{k_{m+1}} = \ldots = V_{k_n} = \mathit{False}$ is a satisfying assignment of the conjunctive normal form

Thus, the conjunctive normal form has a satisfying assignment *if and only if* the plan $\overline{\overline{\Pi}}$ has a legal proper subplan that achieves the goal, in other words, if and only if $\overline{\overline{\Pi}}$ is not perfectly justified. □

Corollary 1 *The problem to find a perfectly justified subplan of a given plan is NP-complete.*

6 Greedy justification

While the task of finding the best possible justified plan is NP-hard, one can design a greedy algorithm that finds an "almost" perfect justification. To check "usefulness" of some operator α in a plan Π, the algorithm proceeds as follows. First, it removes an operator α from the plan. After α has been removed, some operators of Π may become *illegal*, which means that now their preconditions are *not* satisfied before their execution. The algorithm removes every operator which is the first illegal operator in at least one linearization of Π. Then the algorithm examines the resulting plan, finds the remaining illegal operators, and again removes all earliest illegal operators. The algorithm repeats this step until the plan becomes legal. If this plan still solves the goal, then the initially removed operator α was not useful, and we say that α is not *greedily justified*.

The description of the algorithm is presented in Table 3c. The set of illegal operators in line 3 may be found in $O(P \cdot |\Pi|^2)$ time. The same time is required for correctness checking in line 6. Finally, computing the set *Earliest_Illegals* in line 4 requires $O(|\Pi|^2)$ time, if the order of operators is represented by a transitively closed graph. The overall running time of the algorithm is $O(P \cdot |\Pi|^3)$.

As an example, consider again the water-boiling plan:

1. Fill a cup with water.
2. Empty the cup.
3. Fill the cup with water again.
4. Put the cup into a microwave.

Suppose we remove *operator 2*. Now *operator 3* is illegal, because we cannot fill a cup which is already full, and it should be removed from the plan. The resulting plan is:

1. Fill a cup with water.

4. Put the cup into a microwave.

which is legal and solves the goal. Thus, *operator 2* in the initial plan is not greedily justified.

If an operator α in a plan is not well-justified, and we use the algorithm *Greedy_Justify_Checking* to check the *usefulness* of α, then α will be removed at the first step of execution. The resultant plan is legal and solves

operators	preconds	effects
α_i (for each $i \in [1 \ldots n]$)	—	$v_i^+, \neg v_i^-$
β_{ij} (for each $V_i \in C_j$)	v_i^+	c_j, p_{ij}
γ_{ij} (for each $\neg V_i \in C_j$)	v_i^-	c_j, p_{ij}
δ	$\neg v_1^-, \neg v_2^-, \ldots, \neg v_n^-$	$v_1^-, v_2^-, \ldots, v_n^-$ and all $\neg p_{ij}$'s

Table 1: Operators in the proof of NP-completeness

the goal. Thus, if an operator is not well-justified, it is not greedily justified either, and therefore greedy justification is stronger than well-justification. Also, the algorithm is able to detect and remove cycles: if a linearly ordered plan contains a cycle $\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_j$, then, while testing usefulness of α_{i+1}, the algorithm will remove α_{i+1}, then α_{i+2}, then α_{i+3}, and so on till α_j, and then it receives a legal subplan that solves the goal.

A plan is greedily justified if all its operators are greedily justified. It follows from the above discussion that such a plan is always well-justified and does not contain cycles. An algorithm that finds a greedily justified subplan of a plan Π may be briefly described as follows

1. for each operator of the plan Π
1a. use *Greedy_Justify_Checking* to check
    if the operator is greedily justified
1b. if it is *not* greedily justified, we receive some correct subplan Π' of Π; then we recursively call the algorithm for Π', to find its greedy justification
2. if all operators are greedily justified, then our plan is greedily justified, and so a greedily justified subplan of the initial plan is found

It may be shown that *Greedy_Justify_Checking* is called at most $|\Pi|^2$ times, and thus the running time of the algorithm is $O(P \cdot |\Pi|^5)$.

The running time may be considerably improved in the case of a linearly ordered plan. The algorithm for this case is shown in Table 3d. To determine whether some operator α is greedily justified, the algorithm removes this operator and executes the remaining operators in order. If an illegal operator is encountered, the algorithm removes this operator and continues to execute the plan. Thus, it removes all illegal operators and receives the final state that the plan achieves with the illegal operators removed. If the goal is not achieved, then the initially removed operator α is greedily justified. On the other hand, if the new plan achieves the goal, than it is an optimized version of the initial plan. Then we apply our algorithm recursively to check if this new, shorter plan is greedily justified. The running time of the algorithm is $O((P+E)|\overline{\Pi}|^2)$, providing the problem domain contains the finite number of literal classes.

7 Conclusion and Open Problems

This paper formalizes the intuition behind "good" partially and linearly ordered plans. Table 2 presents different kinds of justification and running time necessary to find justified subplan of a plan for each kind of justification. Running time is presented for algorithms dealing with partially ordered sets. Recall that the algorithm to find a greedily justified version of a linearly ordered plan is much faster; it takes only $O((P+E) \cdot |\overline{\Pi}|^2)$ time.

The table may be viewed as a spectrum of justified plans. On one end of the spectrum plans are backward justified. A backward justified subplan of a given plan is not hard to find, but it may contain some "useless" operators. The other end of the spectrum contains perfectly justified plans. They cannot have any useless operators, but it is NP-hard to find a perfectly justified subplan of a given plan.

The results of this paper may be used for creating abstraction hierarchies. According to the definition of *ordered abstraction hierarchies* presented in [Knoblock *et al.*, 1991], different kinds of justification give rise to different ordered hierarchies. *More* restrictive kinds of justifications give rise to *less* restrictive conditions for building an abstraction hierarchy, resulting in finer-grained hierarchies. So, using the definitions of well-justified and greedily justified plans, we may build finer ordered abstraction hierarchies than those generated by Knoblock's ALPINE. The theoretical results and algorithms that allow us to build such finer hierarchies are presented in [Fink, 1992].

References

[Bacchus and Yang, 1991] Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Proceedings of the 12th IJCAI*, pages 286–292, Sydney, Australia, August 1991.

[Fink, 1992] Eugene Fink. Justified plans and ordered hierarchies. Master's thesis, University of Waterloo, Department of Computer Science, Waterloo, Ont., Canada, Forthcoming 1992.

[Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

kind of subplan	**running time**	
perfectly justified	NP-complete	stronger justification
greedily justified	$O(P \cdot \|\Pi\|^5)$	$\uparrow$
well-justified	$O(P \cdot \|\Pi\|^4)$	$\downarrow$
backward justified	$O(E \cdot \|\Pi\|^2)$	weaker justification

Table 2: Kinds of justified subplans and running time to find them

[Knoblock *et al.*, 1991] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the 9th AAAI*, Anaheim, CA, 1991.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

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

[Tenenberg, 1988] Josh Tenenberg. *Abstraction in Planning.* PhD thesis, University of Rochester, Dept. of Computer Science, Rochester, NY, May 1988.

[Tenenberg and Yang, 1990] Josh Tenenberg and Qiang Yang. ABTWEAK: abstracting a nonlinear, least commitment planner. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

Table 3: ALGORIHMS

Backward_Justification
1. let $\overline{\Pi}$ be some linearization of Π;
2. **for** $\alpha :=$ (last operator of $\overline{\Pi}$) **downto** (first operator of $\overline{\Pi}$) **do**
   begin
3. *Justified* := *False*;
4. **for** each $l \in \mathit{Eff}(\alpha)$ **do**
5. **if** $(\exists \alpha_1 \in \Pi$ sth α establishes l for $\alpha_1)$ **or** (α establishes l for S_g)
6. **then** /* α is backward justified */ *Justified* := *True*;
7. **if** *Justified*=*False* /* α is not backward justified */
8. **then** remove α from the plan Π;
   end

(a) Finding the backward justified subplan of a given plan.

Well_Justification
1. **repeat**
2. **for** each $\alpha \in \Pi$ **do**
3. **if** Π without α is legal and achieves the goal
4. **then** remove α from Π
5. **until** no operator is removed during the last execution of the loop

(b) Finding a well-justified subplan of a given plan.

Greedy_Justify_Checking(Π,α)
1. remove α from Π
2. **repeat**
3. *Illegals* := "the set of illegal operators of Π";
4. *Earliest_Illegals* := $\{\alpha' \in \mathit{Illegals} \mid (\forall \alpha_1 \in \Pi)\ \alpha_1 \prec \alpha' \Rightarrow \alpha_1 \notin \mathit{Illegals}\}$
   /* That is *Earliest_Illegals* is the set of earliest illegal operators */;
5. remove all operators of the set *Earliest_Illegals* from Π;
6. **until** Π does not contain illegal operators;
7. **if** Π still achieves the goal
8. **then** return("Π is a legal subplan of the initial plan")
9. **else** return("α in the initial plan is greedily justified")

(c) Checking if the operator α in the plan Π is greedily justified.

Linear_Greedy_Justification($\overline{\Pi}, S_0, S_g$)
1. **for** each $\alpha \in \overline{\Pi}$ **do**
   begin
2. $\overline{\Pi}_1 := \overline{\Pi}$ with α removed;
3. $S := S_0$;
4. **for** $\alpha_1 :=$ (first operator of $\overline{\Pi}_1$) **to** (last operator of $\overline{\Pi}_1$) **do**
5. **if** $\mathit{Pre}(\alpha_1) \subseteq S$ /* α_1 is legal */
6. **then** /* execute α_1 */ $S :=$ state received by applying α_1 to S
7. **else** remove α_1 from $\overline{\Pi}_1$;
8. **if** $S_g \subseteq S$ /* $\overline{\Pi}_1$ achieves the goal S_g */
9. **then** return(Linear_Well_Justification($\overline{\Pi}_1, S_0, S_g$))
   end;
10. return($\overline{\Pi}$)

(d) Finding a greedily justified subplan of a linearly ordered plan.