Formalizing planning knowledge for hierarchical planning

QIANG YANG

Department of Computer Science, University of Waterloo, Waterloo, Ont., Canada N2L 3G1

Received October 19, 1989

Revision accepted March 5, 1990

A hierarchical planning system achieves efficiency by planning with the most important conditions first, and considering details later in the planning process. Few attempts have been made to formalize the structure of the planning knowledge for hierarchical planning. For a given domain, there is usually more than one way to define its planning knowledge. Some of the definitions can lead to efficient planning, while others may not. In this paper, we provide a set of restrictions which defines the relationships between a non-primitive action and its set of subactions. When satisfied, these restrictions guarantee improved efficiency for hierarchical planning. One important feature of these restrictions is that they are syntactic and therefore do not depend on the particular structure of any plan. Along with these restrictions, we also provide algorithms for preprocessing the planning knowledge of a hierarchical planner. When used during planning, the preprocessed operator hierarchies can enable a planner to significantly reduce its search space.

Key words: automated reasoning, problem solving, hierarchical planning, search control, knowledge representation.

Un système de planification hiérarchique permet d'obtenir un rendement efficace dans la mesure où il planifie d'abord les conditions importantes avant de s'attarder aux détails secondaires dans le processus de planification. Quelques tentatives ont été réalisées en vue de formaliser la structure des connaissances de planification dans la planification hiérarchique. Dans un domaine donné, il existe normalement plus d'une façon de définir les connaissances de planification. Certaines définitions peuvent permettre une planification efficace et d'autres non. Dans cet article, l'auteur propose une série de restrictions qui définissent les rapports entre une action non primitive et son ensemble de sous-actions. Lorsqu'elles sont respectées, ces restrictions permettent d'améliorer l'efficacité dans la planification hiérarchique. L'une des caractéristiques importantes de ces restrictions est qu'elles sont syntaxiques et ne dépendent donc pas de la structure particulière d'un plan. En plus de ces restrictions, l'auteur propose également des algorithmes pour le prétraitement des connaissances de planification d'un planificateur hiérarchique. Quand elles sont utilisées durant la planification, les hiérarchies d'opérateur peuvent permettre au planificateur de réduire considérablement son espace de recherche.

Mots clés : raisonnement automatisé, résolution de problèmes, planification hiérarchique, contrôle de recherche, représentation des connaissances.

[Traduit par la revue]

Comput. Intell. 6, 12-24 (1990)

1. Introduction

Hierarchical planning using action reduction is one of the most widely used planning methods. Given a set of high-level actions to be carried out, this method will find ways for reducing each action into subactions according to a predefined set of action reduction schemata. It then resolves possible conflicts among the subactions in the plan using the least-commitment strategy. The process is repeated until all of the actions in the plan are primitive, and all of the interactions between the actions are removed. The advantage of this planning method is that a hierarchical planner works on a small but important set of interactions first, before it sets out to handle the rest which are considered as mere details. This technique has been used in a number of planning systems, including NOAH (Sacerdoti 1977), NONLIN (Tate 1977), and SIPE (Wilkins 1984, 1988).

The domain knowledge of a hierarchical planner is organized hierarchically in the form of action reduction schemata. In addition to a set of primitive actions, a hierarchical planner also has a set of non-primitive actions, as well as a set of action reduction schemata that defines the relationship between the actions. The efficiency of a hierarchical planner depends on how the action reduction schemata are defined; if one is not careful in the definition, one may lose the efficiency of hierarchical planning. Usually, there are several ways of defining the action reduction schemata for a given domain. Some of the definitions will result in improved efficiency, while others will not. The purpose of this paper is to develop restrictions over the definitions of action reduction schemata, so that whenever these restrictions are satisfied, efficiency can be greatly improved.

In particular, if a planner faces conflicts in a plan that are impossible to resolve by ordering the actions, it generally has to try to reduce its actions further in order to resolve the conflicts by interleaving the subactions. This greatly reduces the efficiency of a planner, since it has to search an extra portion of its search space, even if there is strong indication that no solution exists in that space. In this paper, we show that a class of domains exists for which the action reduction schemata can be defined in certain ways, so that unresolvable conflicts in a plan cannot be resolved in any reduction of the plan. We provide a set of restrictions on the schemata definition, and show that when they are satisfied, dead ends in a planner's search space can be detected early in the planning process.

Consider the following example. Suppose one has two goals to achieve, namely, to paint the ceiling and to paint the floor. Assume a hierarchical planner has come up with a plan as shown in Fig. 1. Assume that if one paints the ceiling first, the paint will drip down and make it impossible to paint the floor next. But if one paints the floor first, one cannot get in the room to paint the ceiling. That is, the action "apply-paint-to-ceiling" deletes a precondition for "apply-paint-to-floor," and the latter also deletes a precondition

Printed in Canada / Imprimé au Canada

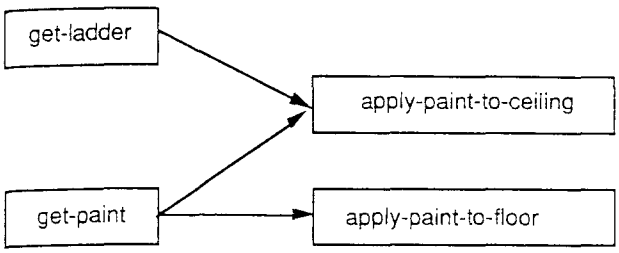


FIG. 1. A plan containing unresolvable conflicts.

for the former. Therefore, the plan in Fig. 1 is unlinearizable. Suppose that the set of action reduction schemata for the painting domain has been marked as satisfying the imposed restrictions, it can *backtrack* from this plan and try other ways to fulfill the goals. If no other way exists, the planner can announce failure. However, if the schemata do not satisfy the property, then the planner cannot rule out reducing the actions further as a means of resolving the conflicts. One such situation will be shown later in Sect. 3.

The results of this paper provide methods for *preprocessing* a given set of action reduction schemata Φ. Given Φ, one can check if every reduction schema in Φ satisfies the restrictions before planning starts. If so, then during planning time the planner can backtrack whenever it has a plan containing unresolvable conflicts. For some domains, not all the action reduction schemata satisfy the restrictions. Thus, we also provide a method for checking whether each individual schema satisfies the restrictions. The ones that satisfy the restrictions are marked as such. As we will see later in the paper, the marked reductions can help improve planning efficiency in a number of ways. The preprocessing process is shown in Fig. 2, where Φ represents the set of given action reduction schemata and Λ represents the set of given action templates.

Equally important for preprocessing, the restrictions also provide a standard for how the reduction schemata should be defined. For a given domain, there are usually several ways of defining the action reduction schemata. If one can define them in such a way that the action templates and the reduction schemata satisfy the restrictions, then planning can be done more efficiently.

2. Defining hierarchical planning systems

This section provides a formal definition of the type of hierarchical planning to be discussed in the paper. In planning research, the term "hierarchical planning" has been used to describe several different types of planning methods, including subgoaling (Fikes and Nilsson 1971), action reduction (Sacerdoti 1977; Tate 1977), and planning at different levels of details (Sacerdoti 1974; Wilkins 1988). To clarify the concept, Wilkins (1986) provides an in-depth discussion of different varieties of planning hierarchies and classifies them into either *planning levels* or *abstraction levels*. A planning level corresponds to the "artifacts of particular planning systems" (Wilkins 1986). For example, a new planning level can be created by expanding each node in a plan according to some predefined action reduction schemata. On the other hand, abstraction levels are distinguished by the "granularity, or the fineness of detail, of the discriminations it makes in the world" (Wilkins 1986). For example, ABSTRIPS (Sacerdoti 1974) is a planner that plans on different levels of abstraction. In this paper, the term "hierarchy" refers to the planning levels that are obtained by reducing actions in a plan using a predefined set of action reduction schemata.

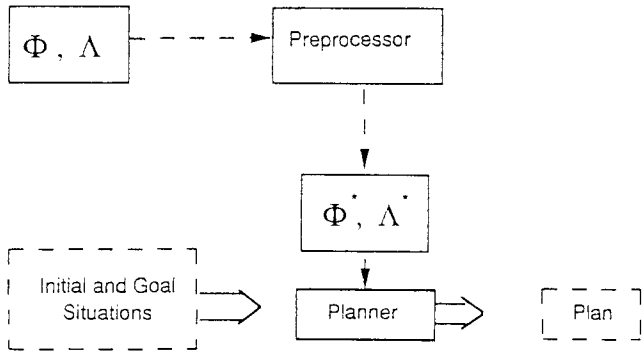


FIG. 2. The preprocessing system configuration.

There are two parts to hierarchical planning with action reduction. The first is the representation of actions and their reductions, and the second is the way problem solving is done using these representations. Below, we provide a formal definition of a hierarchical planning system, including action templates and instances, reduction schemata, and composite reductions of plans.

2.1. Representation

The planning knowledge of a hierarchical planning system consists of two parts. The first part is a set of action templates, Λ. Each action template $\mathbf{a} \in \Lambda$ is represented in terms of preconditions and effects. Specifically, each action template **a** has a set of preconditions, preconditions(**a**), and a set of effects, effects(**a**), where each set consists of literals in first-order predicate logic. Thus, for example, one can choose to represent the action template for moving a block x from the top of another block y to the table in the blocks-world domain as

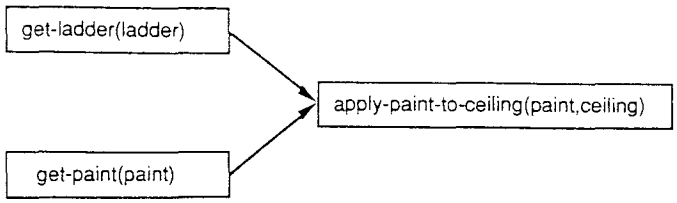
put-block-on-table(x y)
comment: *move x from the top of y to the table.*
preconditions: = {Block(x), Block(y), Cleartop(x), On(x, y)}
effects = {Ontable(x), Cleartop(y), $\neg$ On(x, y)}

An action template in Λ can be *instantiated* by replacing some of the variables in its preconditions or effects by terms. The replacement can be represented by a set of ordered pairs, where the first element of each pair is a variable and the second element is a term. Each such set is called a *substitution*. If **a** is an element of Λ and

$$\beta = \{\langle v_1, t_1\rangle, \langle v_2, t_2\rangle, \ldots, \langle v_n, t_n\rangle\}$$

is a substitution, then $\mathbf{a}\beta$ is **a** with every variable v_i that appears in its preconditions and effects replaced by t_i. $a = \mathbf{a}\beta$ is called an *instance* of **a** and is also referred to as an *action*. In order to make a clear distinction between an action template and an action, the former is denoted in boldface, while the latter is not. If a is an instance of an action template **a**, then the latter is called a *template* of the former, and template(a) = **a**. For simplicity, we assume that each action has a unique template. The results presented below can be easily extended to cases in which there is more than one template for each action instance.

The second component of the planning knowledge of a hierarchical planner is a set of *action reduction schemata* Φ.



<Have(ladder), get-ladder(ladder), apply-paint-to-ceiling(paint, ceiling)>
<Have(paint), get-paint(paint), apply-paint-to-ceiling(paint, ceiling)>

FIG. 3. An example of a schema.

A reduction schema $R \in \Phi$ is a function which, when applied to an action template in Λ, returns a partially ordered set of actions a_i. R is not necessarily applicable to every action template in Λ, and an action template can have more than one reduction schema applicable to it. The set of action reduction schemata applicable to $\mathbf{a}$ is denoted by $\alpha(\mathbf{a})$. $\mathbf{a}$ is *primitive* if $\alpha(\mathbf{a}) = \emptyset$, otherwise it is *non-primitive*. Intuitively, a primitive action template is one that cannot be decomposed further into more detailed steps, while a non-primitive one can.

If R is applicable to $\mathbf{a}$, then $R(\mathbf{a}) = \langle A, E, C \rangle$, where A is a set of actions, E defines a partial ordering among the elements of A, and C is a set of conditions along with specifications of where they must hold. $R(\mathbf{a})$ is called a *reduction* of $\mathbf{a}$. Each element of C is called a *protection interval* (Charniak and McDermott 1984), which is a triple $\langle p, a_1, a_2 \rangle$, where p is a condition, and a_1 and a_2 are actions in A such that (1) $p \in \text{effects}(a_1)$, (2) $p \in \text{preconditions}(a_2)$, and (3) p is expected to be true after a_1 and before a_2. $\mathbf{a}_1$ is called a *provider* of p for $\mathbf{a}_2$.

Let $A(R(\mathbf{a}))$ be the set of actions in $R(\mathbf{a})$. In the following discussion, $a \prec b$ means that a is constrained to occur before b, and $a \prec_{\text{im}} b$ denotes that a is immediately before b. The actions in $A(R(\mathbf{a}))$ have to obey the following restrictions:

1. $\forall e \in \text{effects}(\mathbf{a})$, $\exists a_i \in A(R(a))$ such that $e \in \text{effetcs}(a_i)$ and $\forall b \in A(R(\mathbf{a}))$, if $\neg e \in \text{effects}(b)$ then $b \prec a_i$. That is, every effect of an action template is asserted by at least one of the actions in its reduction.

2. $\forall p \in \text{preconditions}(\mathbf{a})$, $\exists a_i \in A(R(\mathbf{a}))$ such that $p \in \text{preconditions}(a_i)$, and $\forall b \in A(R(\mathbf{a}))$, if $p \in \text{effetcs}(b)$ then $a_i \prec b$. That is, every precondition of an action template is also a precondition of at least one of the actions in its reduction, and further this precondition is not the effect of some earlier subaction.

3. $\forall a_i \in A(R(\mathbf{a}))$, $\forall p \in \text{preconditions}(a_i)$, $\forall b \in A(R(\mathbf{a}))$ such that $a_i \not\prec b$, if $\neg p \in \text{effects}(b)$ then $\exists c \in A(R(a))$ such that $c \prec a_i$ and $b \prec c$, and $p \in \text{effects}(c)$. In other words, a reduction is a miniature plan free of any conflicts.

The definition of reduction schemata above is intended to capture the formal aspects of action or goal expansions in a number of systems. In particular, a reduction schema R corresponds to a "soup code" in NOAH (Sacerdoti 1977), an "opschema" or an "actschema" in NONLIN (Tate 1977), and a "plot" in SIPE (Wilkins 1984). However, several simplifications are made in our formalization of reduction schemata, the first being that "reduction assumptions" are not included explicitly in our definitions. A reduction assumption[1] (Charniak and McDermott 1984) is a condition on the applicability of a reduction. For example, to clear the top of a block x, a planner may choose a particular reduction that involves two steps: clear the top of block y that is on top of x, then move y to the table. But this reduction is needed only if x is not already clear on its top. In this case, $\text{On}(y, x)$ is a reduction assumption, and an interval is set up that protects the assumption if the reduction is selected. Reduction assumptions are the preconditions of certain subactions in a reduction R that are not achieved by some other subactions in R, and are used by the control component of a planner to restrict the use of certain reductions. The state space generated by all possible reductions that include reduction assumptions is a subset of the state space defined by our formalization. Since the results to be presented below concern the nonexistence of solutions in a portion of a state space, it will be easy to verify that all our subsequent results will hold with the inclusion of reduction assumptions. Thus, we omit them for simplicity.

The second simplification is that no distinction is made between a goal and an action in our formalization; all goals are represented as action templates in Λ which can be reduced to a special action **no-op**, a primitive action which means that the action can be achieved by doing nothing. With this definition, a goal can be achieved in two ways. One is to reduce it to no-op, and impose precedence and variable binding constraints so that the goal holds. Another is to reduce it using a reduction schema that represents a partially ordered plan, so that when the subactions in the schema are executed in an order consistent with the partial order, the goal will be achieved. These two ways for achieving a goal are equivalent to the traditional view of goals as situations that can be reached using actions.

Let $a = \mathbf{a}\beta$ be an instance of $\mathbf{a} \in \Lambda$. We extend the definition of action reduction schemata to instances of action templates, as follows: $\forall R \in \alpha(\mathbf{a})$, $R(a) = R(\mathbf{a})\beta$, where $R(\mathbf{a})\beta$ is the result of applying the substitution β to all the actions in $A(R(\mathbf{a}))$, and to all the action occurrences in $C(R(\mathbf{a}))$. If R is applicable to a, then $R(a)$ is called a *reduction* of a, and each action in $R(a)$ is called a *subaction* of a. Intuitively, a reduction for an instance a of $\mathbf{a}$ consists of a partially ordered set of actions, each of which is an instance of an action template in $A(R(\mathbf{a}))$. Extended in this way, it is clear that the following properties hold: Let $a = \mathbf{a}\beta$. Then,

1. $\alpha(a) = \alpha(\mathbf{a})$, and
2. a is primitive if and only if $\mathbf{a}$ is.

Figure 3 describes an action reduction schema for painting a ceiling. A set of protection intervals is also given in the figure. The preconditions and effects of each action are given in Table 1.

2.2. Planning with action reductions

A plan is defined as a partially ordered set of actions. Let P be a plan, then $P = \langle A, E, C \rangle$, where A is a set of actions, E defines a partial ordering among the actions in A, and C is a set of protection invervals. Each action in A is an instance of some action template in Λ.

Let a be an action in a plan P, and R be a reduction schema applicable to a. Then $R(P)$ is the plan that results

[1] It is also called an "use-when" condition in NONLIN (Tate 1977).

TABLE 1. Actions in the painting example with their preconditions and effects

Action	Preconditions	Effects
paint-ceiling(*ceiling*)	∅	{Painted(*ceiling*)}
get-ladder(*ladder*)	∅	{Have(*ladder*)}
get-paint(*paint*)	∅	{Have(*paint*)}
apply-paint-to-ceiling(*paint*, *ceiling*)	{Have(*ladder*), Have(*paint*), Climbable(*ladder*)}	{Painted(*ceiling*), ...}

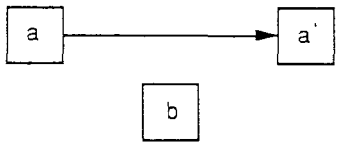


FIG. 4. A deleted-condition conflict.

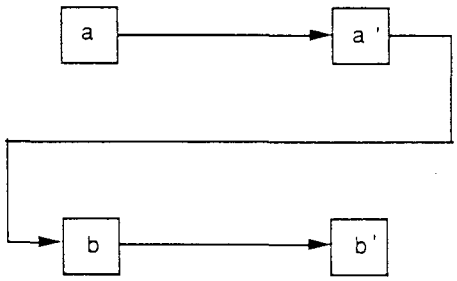


FIG. 5. Imposing an ordering constraint.

when a is replaced by $R(a)$ in P. Let $\delta = \langle p, a, b \rangle$ be a protection interval in P. We will now define how to transpose a protection interval to a reduction of a plan containing the interval.

After P is reduced using R, δ is no longer associated with $R(P)$, instead $R(P)$ will have one or more protection intervals derived from δ which involve subactions $a_i \in A(R(a))$. In particular, let a_i be a subaction of a such that

1. $p \in$ effects(a_i). That is, a_i asserts p, and
2. $\forall a_j \not\prec a_i$, $\neg p \notin$ effects(a_j). That is, p is not denied by any other subactions after a_i.

By the definition of R, one or more such a_i always exists. Then $\langle p, a_i, b \rangle$ is a protection interval associated with $R(P)$.

Similarly, let b be an action in a plan P, and R' be a reduction schema applicable to b. Let $R'(P)$ be the plan that results when b is replaced by $R'(b)$ in P. Suppose $\langle p, a, b \rangle$ is a protection interval in P. Then $R'(P)$ is associated with one or more protection invervals involving the subactions of b. Let b_i be a subaction in $A(R'(b))$, satisfying the following conditions:

1. $p \in$ preconditions(b_i). That is, p is a precondition of b_i, and
2. $\forall b_j \in A(R(b))$, if $p \in$ effects(b_j) then $b_i \prec b_j$. That is, no other subactions of b can establish p for b_i.

Then $\langle p, a, b_i \rangle$ is a protection interval associated with $R'(P)$.

The above definition for interval and plan reduction in one step can be naturally extended to reduction in more than one step. Let P be a plan, and $R_1, R_2, \ldots, R_n$ be reduction schemata. Then

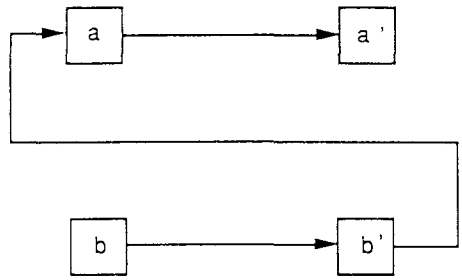


FIG. 6. A different ordering.

$$Q(P) = R_1(R_2(\ldots(R_n(P))\ldots)$$

is a *composite reduction* of P. Furthermore, if Δ is a set of protection intervals associated with P, then $Q(\Delta)$ is the corresponding set of protection intervals associated with $Q(P)$.

A hierachical planner starts planning for a given set of goals by finding appropriate actions to achieve them. If $G(X)$ is a goal, then the action chosen to achieve this goal should have $G(x)$ as one of its effects. These actions form a plan at the highest level. The subsequent problem solving process can be described in the following steps:

1. Choose a non-primitive action and replace it by one of its reductions. Let the new plan be P.
2. Find out the set of interactions among the actions in P, and suggest ways to handle them.
3. If all the actions in P are primitive, then terminate planning. Else go to step 1.

After step 1 is done, certain new interactions may appear. Depending on the particular domain of application, interactions can be of different types. The only type of interaction considered in this paper is *deleted-condition conflict* between a pair of actions. This type of interaction occurs when an action in a plan deletes a precondition or an intended effect of some other action. More formally, let P be a plan and $\langle p, a, a' \rangle$ be a protection interval in P. Suppose there is some action b in P such that $\neg p \in$ effects(b). If $b \not\prec a$ and $a' \not\prec b$, then a deleted-condition conflict occurs in the plan P (see Fig. 4). This conflict can be characterized according to whether $\neg p$ is also an intended effect of b:

1. Both $\langle p, a, a' \rangle$ and $\langle \neg p, b, b' \rangle$ are protected intervals in P, so the conflict is denoted by the pair $(\langle p, a, a' \rangle, \langle \neg p, b, b' \rangle)$. In this case, one way to remove the conflict is to impose a time ordering such that a' occurs before b (Fig. 5), or b' occurs before a (Fig. 6).

2. If $\neg p$ is not needed by any action in P, then it is not a protected condition, so the conflict can be denoted by the pair $(\langle p, a, a' \rangle, \langle \neg p, b \rangle)$. In this case, one way to remove the conflict is to order b before a (Fig. 7) or after a' (Fig. 8).

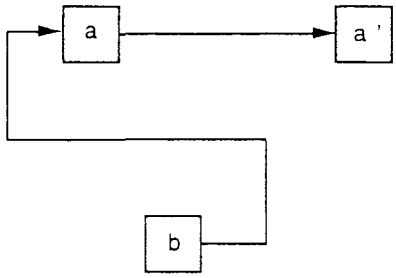


FIG. 7. Another ordering constraint.

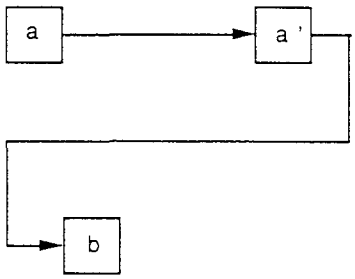


FIG. 8. Last way of taking care of the conflict.

3. Motivation

A hierarchical planner plans by repeatedly selecting and reducing the non-primitive actions, and applying critics to detect and resolve conflicts in a plan. Sometimes the critics may find conflicts in a plan that are impossible to resolve. A plan in this situation is said to have *unresolvable conflicts*. A number of planners such as NONLIN (Tate 1977) or SIPE (Wilkins 1984) will *backtrack* from such a plan to try other ways to achieve the goals, or announce failure if no alternative ways exist. However, this is not always the correct behavior, because there exist situations in which conflicts that are unresolvable in a plan may be resolvable in a composite reduction of the plan.

As an example of this, suppose a plan contains two higher-level actions a and b, with no constraints on their relative ordering (Fig. 9a). Suppose that their effects and preconditions are

$$\text{preconditions}(a) = \{x\}, \quad \text{effects}(a) = \{u, \neg y\}$$
$$\text{preconditions}(b) = \{y\}, \quad \text{effects}(b) = \{w, \neg x\}$$

where the propositions u, w, x, and y are all distinct. Then the following conflicts will occur: an effect of a deletes a precondition of b, and an effect of b deletes a precondition of a. This kind of conflict is often called a "double-cross," meaning that neither possible ordering of a and b will work. Unless a planning system can resolve this conflict by inserting additional actions between a and b to restore the needed preconditions, it will normally announce failure. However, suppose that

1. action a can be reduced to the subactions $a_1 \prec a_2$, with preconditions(a_1) = preconditions(a), effects(a_2) = effects(a), and $x \notin$ preconditions(a_2); and
2. b can be reduced to $b_1 \prec b_2$, with preconditions(b_1) = preconditions(b) and effects(b_2) = effects(b), and $y \notin$ preconditions(b_2).

Then the interaction can be resolved in the reduced plan by assigning orderings such that $a_1 \prec b_2$ and $b_1 \prec a_2$ (Fig. 9b).

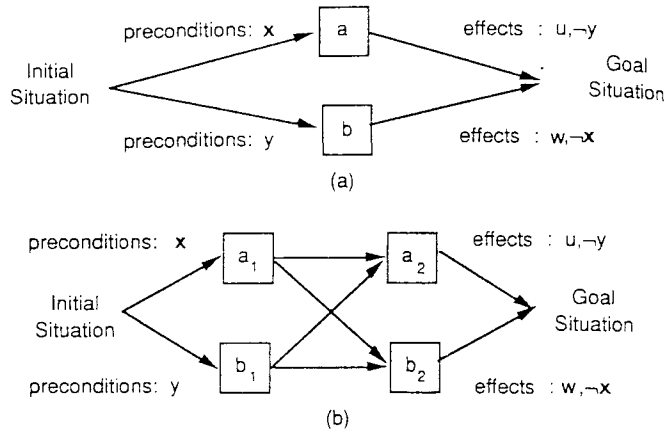


FIG. 9. (a) A plan with unresolvable conflicts; (b) resolving the conflicts by reducing the plan in (a).

In the above example, a conflict that appears unresolvable at a higher level is in fact resolvable at a lower level. Therefore, a planning system should consider reducing a plan as a way to resolve unresolvable conflicts.

However, it is undesirable to do this for the following two reasons. First, an unresolvable conflict usually indicates that it is unlikely that solution will be found, no matter how the solution is refined. It would be desirable for a planner to have the property that whenever a solution exists at a lower level of hierarchy, the high-level version of that solution is also correct. Tenenberg (1988) called such a property the *upward-solution property* for ABSTRIPS type of hierarchical planning, in which a higher-level action can be created by eliminating conditions of a low-level action, which are considered as less important. By imposing certain syntactic restrictions on how the ABSTRIPS abstraction hierarchy is defined, Tenenberg proved that the upward-solution property holds. One purpose of this paper is to ensure that a similar property holds across planning levels as well. Second, the necessity for a planner to search through such branches in its search space makes planning more inefficient. If one may ensure that a planner can backtrack whenever unresolvable conflicts occur, and that this can be done without losing any possible solutions, planning efficiency can be improved. For certain domains, such a property can be enforced by imposing a set of syntactic restrictions on how the action reduction schemata should be defined. For others, such a property is not satisfied by all of the actions, but the syntactic restrictions could enable one to preprocess the set of action reduction schemata so that all the actions that satisfy the restrictions can be identified a priori.

In the following sections, we will first define unlinearizability of a plan, and design restrictions to be imposed on reduction schemata so that unresolvable conflicts in a plan indicate that the conflicts cannot be resolved in any composite reduction of the plan.

4. Plan unlinearizability

A partially ordered plan has many possible *linearizations*, each being a totally ordered sequence of actions in the plan. If a and b are actions in a plan P and L is a linearization of P, then $a \prec b$ in L if $a \prec b$ in P. A plan can also have different instantiations of its variables. Any instantiated and linearized plan that is free of conflicts can be viewed as an instance of the original plan and can be used to achieve the

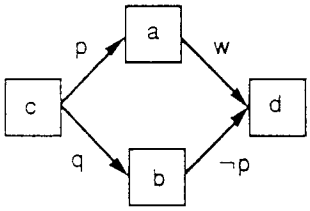


FIG. 10. A plan with four actions a, b, c, and d, along with protection intervals in the plan.

given goals. In the following, the word "linearization" of a plan refers to any "ground" linearization of the plan, where all the terms in a substitution β are constants.

Protection-interval violation is a situation when an action denying a condition p is in the middle of an interval in which p is protected. Formally,

Definition 4.1

Let $\delta = \langle p, a, b \rangle$ be a protection interval in a plan P. δ is *violated* in the plan if and only if $\exists c \in A(P)$ such that $a \prec c$, $c \prec b$, and $\neg p \in \text{effects}(c)$.

In general, a plan may contain several deleted-condition conflicts. For such a plan, we would like to know whether or not a linearization exists in which none of the protection intervals is violated.

Definition 4.2

Let P be a plan and Δ be a set of protection intervals in P. P is said to be Δ-*linearizable* if and only if there exists a linearization L of P such that $\forall \delta \in \Delta$, δ is not violated in L. If L exists, it is said to be Δ-consistent.

For example, for the plan in Fig. 10, the set of all protection intervals is

$$\Delta = \{\langle p, c, a \rangle, \langle w, a, d \rangle, \langle q, c, b \rangle, \langle \neg p, b, d \rangle\}$$

For this plan, the linearization $c \prec a \prec b \prec d$ is Δ-consistent.

A linearization of a plan is Δ-*inconsistent* if it is not Δ-consistent. A plan P is Δ-*unlinearizable* if and only if every linearization of P is Δ-inconsistent. If Δ contains a set of protection intervals in a plan P, and if P Δ-unlinearizable, then P is said to contain "unresolvable conflicts." For example, the plan in Fig. 9*a* contains unresolvable conflicts, since it is Δ-unlinearizable for

$$\Delta = \{\langle x, i, a \rangle, \langle y, i, b \rangle\}$$

where i represents the initial situation.

When a plan P is reduced in one or more steps to $Q(P)$, the set of protection intervals associated with $Q(P)$ is the reduction of the intervals in P, plus a set of protection intervals associated with the reductions in Q. More formally, let

$$Q(P) = R_1(R_2(\ \ldots\ (R_n(P))\ \ldots)$$

be a composite reduction of P. Also let Δ be a set of protection intervals associated with P, and $Q(\Delta)$ be the corresponding composite reduction of Δ. Let $C(Q)$ be the set of protection intervals associated with the reduction schemata R_i, $i = 1, 2, \ldots, n$. Then the *augmented composite reduction* of Δ is

$$\text{AU}_Q(\Delta) = C(Q) \cup Q(\Delta)$$

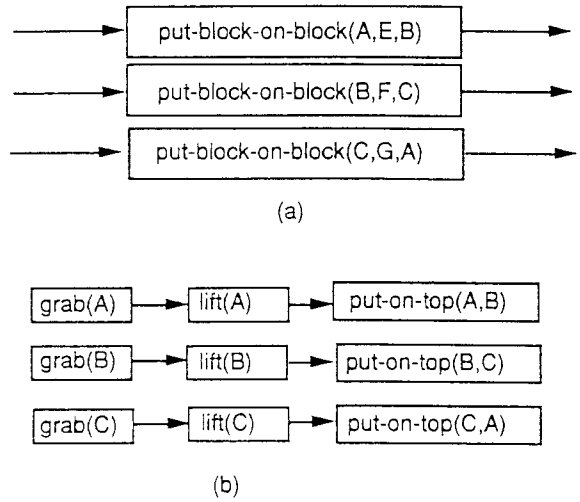


FIG. 11. An example of augmented protection interval.

We would like a set of reduction schemata Φ to have the following property: an unlinearizable plan remains unlinearizable no matter how it is reduced using the reduction schemata in Φ. This property is called "downward-unlinearizable." More formally,

Definition 4.3

A set of action reduction schemata Φ is *downward-unlinearizable* if and only if the following condition holds. Let P be a plan and Δ be a set of protection intervals in P. Also let $Q(P)$ be any composite reduction of P using the action reduction schemata in Φ, and $\text{AU}_Q(\Delta)$ be the corresponding augmented composite reduction of Δ. Then if P is Δ-unlinearizable then $Q(P)$ is $\text{AU}_Q(\Delta)$-unlinearizable.

Consider a blocks-world example where there are three parallel actions: put-block-on-block(A, B), put-block-on-block(B, C), and put-block-on-block(C, A) (see Fig. 11*a*). Assume that three robot hands are available, which are designated as H_1, H_2, and H_3. Let i be the initial situation, and

$$\begin{aligned}\Delta = \{&\langle \text{Cleartop}(A),\ i,\ \text{put-block-on-block}(A,\ E,\ B)\rangle,\\ &\langle \text{Cleartop}(B),\ i,\ \text{put-block-on-block}(B,\ F,\ C)\rangle,\\ &\langle \text{Cleartop}(C),\ i,\ \text{put-block-on-block}(C,\ G,\ A)\rangle\}\end{aligned}$$

Let the plan in Fig. 11*a* be P. Then P is Δ-unlinearizable.

Now consider the composite reduction $Q(P)$ of P in Fig. 11*b*, with

$$\begin{aligned}Q(\Delta) = \{&\langle \text{Cleartop}(A),\ i,\ \text{grab}(A)\rangle,\\ &\langle \text{Cleartop}(B),\ i,\ \text{grab}(B)\rangle,\\ &\langle \text{Cleartop}(C),\ i,\ \text{grab}(C)\rangle\}\end{aligned}$$

$Q(P)$ is not linearizable with respect to the set of all protection intervals associated with it. This is because $Q(P)$ is also associated with

$$\begin{aligned}\{&\langle \text{Holding}(H_1,\ A),\ \text{grab}(A),\ \text{put-on-top}(A,\ B)\rangle,\\ &\langle \text{Holding}(H_2,\ B),\ \text{grab}(B),\ \text{put-on-top}(B,\ C)\rangle,\\ &\langle \text{Holding}(H_3,\ C),\ \text{grab}(C),\ \text{put-on-top}(C,\ A)\rangle\}\end{aligned}$$

Let $\text{AU}_Q(\Delta)$ be the union of $Q(\Delta)$ and the above set of protection intervals. Since a robot cannot hold a block when another block is on top of it, it is not hard to see that $Q(P)$ is $\text{AU}_Q(\Delta)$-unlinearizable.

Our method for checking if a given set of reduction schemata is downward-unlinearizable is to develop a set of sufficient syntactic restrictions upon action templates in Λ and reduction schemata in Φ. In the next section, we first provide a set of restrictions which guarantees the downward-unlinearizability property for a set of reduction schemata. Later in this paper we will discuss how to simplify the restrictions so that preprocessing can be done efficiently.

5. Imposing syntactic restrictions

In this section, a set of restrictions is provided which restricts the way an action relates to its set of subactions in its reduction. These restrictions are intended to enable one to check and possibly modify a given set of reduction schemata. Whenever the restrictions are satisfied, backtracking from a plan with a set of unresolvable conflicts will not lose any potential solutions at any level of reduction below.

The restriction to be presented concerns conditions under which an action b cannot be interleaved with the subactions of a. To formally state the restrictions, we first define a number of useful concepts. Let a be an action and p a literal. *a deletes p* if $\exists q \in$ effects(a) such that $\neg q$ is unified with p.

Definition 5.1

b conflicts with $R(a)$ if $\forall a_i$, a_j such that $a_i \prec_{\text{im}} a_j$ or a_i is unordered with a_j *in* $R(a)$, any of the following conditions is true:

1. $\exists a_k \in A(R(a))$, $a_k \prec a_i$ or $a_k = a_i$ and a_k deletes $q \in$ preconditions(b). That is, some subaction of a deletes a precondition of b.
2. $\exists a_{i'}$, $a_{j'} \in A(R(a))$, where $a_{i'} \prec a_i$ or $a_{i'} = a_i$, and $a_j \not\prec a_{j'}$ or $a_j = a_{j'}$, such that for some p, $\langle p, a_{i'}, a_{j'} \rangle \in C(R(a))$ and b deletes p. That is, b deletes a condition protected in the reduction of a.
3. $\exists a_l \in A(R(a))$, $a_j \prec a_l$ or $a_j = a_l$, $H =$ (preconditions(a_l) $\cap$ preconditions(a)) $\neq \emptyset$, and for some $p \in H$, b deletes p. That is, b deletes a precondition of some subaction of a.

The above three cases enumerate the conditions when b cannot be interleaved with the subactions of a. If b conflicts with $R(a)$, then b cannot be between any pairs of subactions of a in a consistent linearization of a plan.

Restriction 5.1

If b deletes p for some $p \in$ preconditions(a), or a deletes q for some $q \in$ effects(b), then b conflicts with $R(a)$.

Restriction 5.2

If a deletes p for some $p \in$ preconditions(b), or b deletes q for some $q \in$ effects(a), then b conflicts with $R(a)$.

Intuitively, Restriction 5.1 says that if b cannot be immediately before a in a consistent linearization of a plan P, then b cannot be interleaved with the subactions of a either. Restriction 5.2 can be likewise interpreted. If either of the above restrictions are satisfied for actions a and b along with a reduction $R(a)$ of a, then we say $R(a)$ satisfies that restriction with respect to (w.r.t) b.

As an example, let a be the action put-block-on-block (x, u, y) in the blocks-world domain, and b be the action put-block-on-block(y, v, x). Then the reduction of a given in Fig. 11*b* satisfies Restriction 5.1 w.r.t. b. This is because the first subaction of a, grab(x), deletes Cleartop(x), which is a precondition of b. Thus, b conflicts with $R(a)$.

If one of the above restrictions is satisfied by all the actions in a plan, then any reduction of an unlinearizable plan remains unlinearizable after it is reduced. This result is stated formally in the following theorem:

Theorem 5.1

Let P be a plan containing a non-primitive action a. Suppose for every action b in P, $R(a)$ satisfies Restriction 5.1 (Restriction 5.2) w.r.t. b. Let $R(P)$ be the reduction of P by reducing a using $R(a)$. Then if P is Δ-unlinearizable, then $R(P)$ is $\text{AU}_R(\Delta)$-unlinearizable.

The proofs for this and subsequent theorems are given in Appendix A.

The above result can be extended to plans that are reduced in any number of reductions, as follows:

Corollary 5.1

Let Φ be a set of action reduction schemata. Suppose $\forall \mathbf{a}, \mathbf{b} \in \Lambda$, $\forall R \in \alpha(\mathbf{a})$, $R(\mathbf{a})$ satisfies Restriction 5.1 (or Restriction 5.2) w.r.t. $\mathbf{b}$. Then Φ is downward-unlinearizable.

The above results can be used in several ways. First, if a plan is unlinearizable with respect to its set of protected intervals, then one can check to see, for the available reductions, if all the reductions satisfy any of the restrictions given above. If satisfied, one can conclude from Theorem 5.1 that the reduction of the plan will remain unlinearizable. This restricts the number of choices left to the planner, thus reducing the search space. Second, if it is possible to check the conditions in Corollary 5.1 a priori, then the restrictions do not need to be checked at planning time. That is, whenever a plan is unlinearizable with respect to its set of protected conditions, no reduction of the plan is linearizable.

However, the conditions for Corollary 5.1 are usually very costly to check for arbitrary reduction schemata. The reason is that it requires checking for every possible substitution, the number of which might be very large. What is needed is a set of conditions that are easy to check beforehand, which still ensures the downward-unlinearizability property. This is the intuition that leads to the next section, in which we will discuss a restriction that is rather easy to check.

6. The Unique-Main-Subaction Restriction

We now use the results of the previous section and propose a simplified restriction. This restriction is more restrictive than the previous one in that it requires every non-primitive action have a *unique* main subaction that is also protected by the preconditions of the higher-level action. However, significant computational cost for preprocessing the planning knowledge can be reduced. Later on in this section, we will examine a number of example domains which satisfy this restriction.

Consider the following restriction:

Restriction 6.1

Let a be an action and R be a reduction schema applicable to a. The actions in $A(R(a))$ satisfy the following conditions: $\exists a_{\text{m}} \in A(R(a))$ such that

1. effects(a) $\subseteq$ effects(a_{m}) and $\forall a_i \in A(R(a))$, if $a_i \neq a_{\text{m}}$, then effects(a_i) $\cap$ effects(a) $= \emptyset$. That is, a_m asserts all the effects of a, and no other subactions of a assert any effect of a.
2. preconditions(a) $\subseteq$ preconditions(a_m), and $\forall a_i \in A(R(a))$ such that $a_i \neq a_m$, if $a_m \not\prec a_i$ then effects(a_i)

TABLE 2. An action reduction schema satisfying the Unique-Main-Subaction Restriction

Action	Preconditions	Effects
fetch	Inroom(object1, room1)	Holding(robot1, object1)
achieve(Inroom(robot1, room1))	∅	Inroom(robot1, room1)
achieve(Nextto(robot1, object1))	∅	Nextto(robot1, object1)
pickup	Inroom(object1, room1) Inroom(robot1, room1) Nextto(robot1, object1)	Holding(robot1, object1)

$\cap$ preconditions(a) = ∅. That is, a_m requires all the preconditions of a, and none of these conditions are achieved by some other subactions of a.

This restriction will be referred to as the *Unique-Main-Subaction* Restriction. Intuitively, this restriction requires that the reduction $R(a)$ of an action a contains a "main" subaction a_m which asserts everything a asserts. In addition, the preconditions of a are required to "persist" till the beginning of a_m. Notice that this restriction is stronger than Restriction 5.1 or 5.2 in the previous section.

A number of planning domains can be represented in ways that satisfy this restriction. For example, consider an action of fetching an object "object1" in a room "room1" (based on Wilkins (1988)). The action "fetch" can be reduced into a sequence of three subactions: getting into room1, getting near object1, and picking up object1, in that order. The third subaction, picking up object1, can be considered as the main subaction in this reduction. Notice that the condition that object1 is in room1 is a precondition of both the non-primitive action, fetch, and the main subaction. The complete reduction schema is given in Table 2.

Let Λ be the set of action templates of a planning system, and Φ be the set of action reduction schemata.

Restriction 6.2

Φ satisfies Restriction 6.2 if and only if for every reduction schema $R \in \Phi$ and every action template $\mathbf{a} \in \Lambda$ to which R is applicable, $R(\mathbf{a})$ satisfies the Unique-Main-Subaction Restriction.

Theorem 6.1

Every set of action reduction schemata satisfying Restriction 6.2 is downward-unlinearizable.

Before discussing how this theorem can be used for checking the downward-unlinearizability property, we would like to comment on the applicability of the Unique-Main-Subaction Restriction to planning domains. Intuitively, the Unique-Main-Subaction Restriction requires that every non-primitive action a has a unique main subaction a_m that asserts every effect of a, and requires every precondition of a as its precondition. Domains that satisfy this restriction have the following characteristics:

1. The goals to be achieved can always be broken down to several less-complicated subgoals to solve. For each subgoal, a number of primitive actions are available for achieving it. Each such action requires several preparation steps before it can be performed, and a number of clean-up steps after it is done. This action can be considered as the main step in a reduction schema.

2. For each group of actions mentioned above, a hierarchy is built by associating with a non-primitive action a set of effects which are the purposes of the main step mentioned above, and a set of preconditions which are certain important preconditions of the main step.

A number of domains can be formulated in ways that satisfy the property of downward-unlinearizability. In Appendix B, we provide a representation of the planning knowledge of the blocks-world domain. In this domain, a number of blocks exist on the top of a table. A robot can move one block on top of another, or onto the table. Each block can have at most one block on top of it, and a block can be at most on one other block. Condition On(x, y) represents that the block x is on top of another block y, Ontable(x) means that the block x is on the table, and Cleartop(x) means that block x has no other blocks on top of x. The top-level actions are also the subgoals that the system knows how to solve. They are achieve (On(x, y)), achieve(Ontable(x)), and achieve(Cleartop(x)). More complicated goals can be composed from the conjunction of these.

It is not hard to verify that every reduction satisfies the Unique-Main-Subaction Restriction. For example, consider the reduction R_1 for achieve(On(x, y)). Notice that in R_1(achieve(On(x, y))), the action makeon-block-1(x, y) is the main subaction, which asserts the effect On(x, y) of achieve On(x, y)). In addition, the set of preconditions of achieve(On(x, y)) is empty, and neither achieve(Cleartop(x)) nor achieve(Cleartop(y)) asserts On(x, y). Thus, the Unique-Main-Subaction Restriction is satisfied. Likewise, the reduction R_5 for makeon-block-1(x, y) also satisfies this restriction, since the reduction contains only one subaction. For the rest of the reductions, the Unique-Main-Subaction Restriction can be similarly verified.

As another example, consider a domain in which there is a room, a ladder, supplies for painting, as well as a robot whose goal is to paint portions of the room and (or) the ladder. Suppose in addition to the actions and reduction schemata for painting, the robot also knows how to fetch an object using the action "fetch" in Table 2. Represented in this way, every action reduction schema satisfies the Unique-Main-Subaction Restriction. For instance, the "apply-paint-to-ceiling" step in the reduction of "Paint (Ceiling)" is the main subaction. Also, the "pickup" step in the reduction of "fetch" is also a main subaction. Moreover, these reduction schemata all satisfy the Unique-Main-Subaction Restriction.

The advantage of the Unique-Main-Subaction Restriction is that it allows for more efficient preprocessing. Specifically, this restriction only restricts the way a non-primitive action relates to its set of subactions. Thus the checking does not have to take into account any other actions in the planner's planning knowledge.

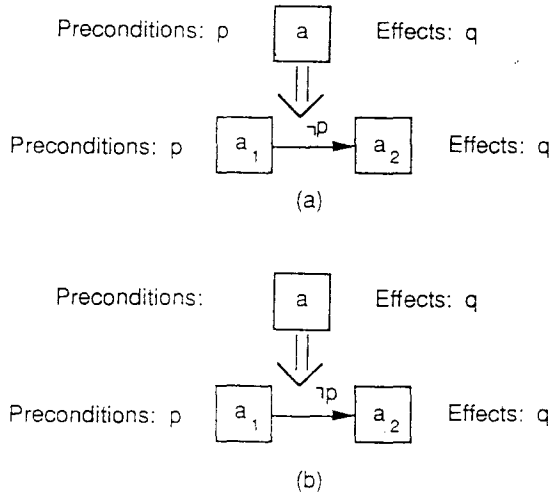


FIG. 12. Modifying a reduction schema (a) by removing some preconditions of an action template (b).

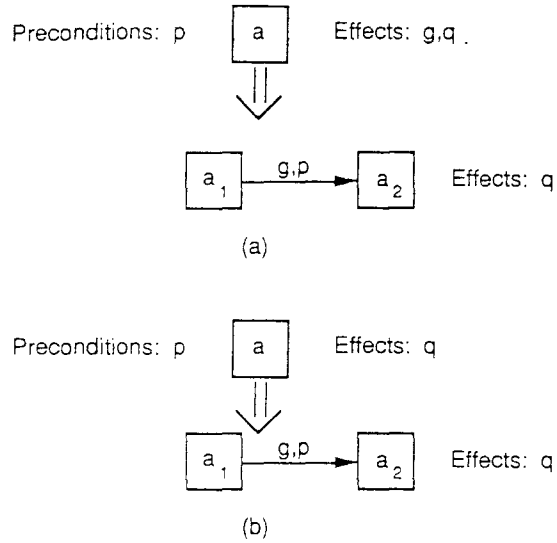


FIG. 13. Modifying a reduction schema (a) by removing effects of action template **a** (b).

7. Checking for downward-unlinearizability

The algorithm for preprocessing a set of action reduction schemata makes use of Restriction 5.1. This algorithm is run after the planning knowledge for a domain is defined, taken as input the set of action reduction schemata, Φ, and action templates, Λ. Two goals will be achieved through preprocessing: (1) to see if the set of action reduction schemata is downward-unresolvable, and (2) if not, try to modify the schemata so that they have this property. Let Φ be a set of action reduction schemata and Λ be a set of actions. Together Φ and Λ provides the planning knowledge for a system. The algorithm for preprocessing is presented below. It returns "true" if Φ satisfies the restrictions imposed in the previous sections. If so, the set of schemata, Φ, satisfies the downward-unlinearizability property.

```
Algorithm Checking.
  begin
    B := Λ;
    while B ≠ ∅ do
      a := pop(B);
      if ∀R ∈ α(a), R(a) satisfies Restriction 6.1 then
      Mark a;
      B := B − {a};
    end; (while)
    if all the action templates in Λ are marked then
      return(True)
    else return(False)
  end
```

To check if $R(\mathbf{a})$ satisfies the Unique-Main-Subaction Restriction, first check to see if there is a unique main subaction in $A(R(\mathbf{a}))$ that asserts every effect **a** asserts and has in its set of preconditions every precondition of **a**. If such a main subaction a_m exists, then check for every other subaction a_i in $A(R(\mathbf{a}))$.

1. $\forall p \in$ preconditions(**a**), $\langle p, a_i, a_m \rangle \notin C(R(\mathbf{a}))$. That is, no other subaction of **a** asserts p for a_m.
2. effects(a_i) $\cap$ effects(**a**) = $\emptyset$. That is, no other subaction of **a** asserts an effect of **a**.

To check each $R(\mathbf{a})$ for unique-main-subaction restriction, the above algorithm suggests a worst-case time complexity of $O(|\Lambda|)$, where $|\Lambda|$ is the number of actions in Λ. Let k be the maximum number of action reduction schemata applicable to an action template. Then Algorithm *Checking* has a worst-case complexity of $O(k \times |\Lambda|^2)$.

If the algorithm *Checking* returns "true," then the set of reduction schemata, Φ, satisfies the downward-unresolvability restriction. In which case, whenever a hierarchical planner detects unresolvable conflicts in a plan, it does not have to consider the reduction of the plan as a means of resolving the conflicts.

8. Modifying action reduction schemata

If the algorithm *Checking* returns "false" for Φ, then in some cases it might be possible to modify the schemata, so that Φ is made to satisfy the restrictions of previous sections.

For example, let **a** be an action template with a precondition p and an effect q. Suppose $R(\mathbf{a})$ consists of two subactions, a_1 followed by a_2, such that the precondition of a_1 is p, and the effect of a_2 is q (see Fig. 12a). If $\langle \neg p, a_1, a_2 \rangle$ is a protection interval associated with $R(\mathbf{a})$, then clearly Restriction 5.1 is violated. One way to modify this reduction schema is to remove the precondition p from effects(**a**). As a result, the modified reduction R' satisfies Restriction 5.1. This new reduction schema is shown in Fig. 12b.

The above example shows how to modify a reduction schema by removing the preconditions of an action template. Other ways of schemata modification also exist. For instance, let **a** be an action template with two effects, q and g, such that the subaction a_1 of **a** has an effect q while another subaction a_2 has an effect g (Fig. 13a). If g is considered as a side effect of **a** that is considered to be less important than q, then removing g from the effects of **a** will make the reduction schema satisfy the Unique-Main-Subaction Restriction. This is shown in Fig. 13b.

Unfortunately, the above examples do not suggest any general procedures for modifying a given set of action reduction schemata. This is because changing the representation of the action templates in Λ may make them less expressive than before. This can occur when, for example, some effects of an action are removed from its effects set. In extreme cases, changing the representation of an action template may

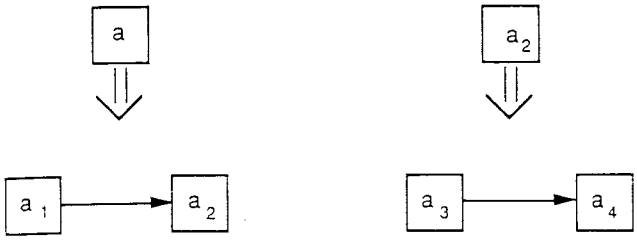


FIG. 14. Reduction schemata.

cause planning to be less efficient than before. For example, removing the preconditions from an action may delay the detection of deleted-condition conflicts with the removed conditions, and as a result, more reasoning may be needed to fix those interactions after the modification. On the other hand, there may exist classes of planning problems for which a particular kind of schemata modification method exists, and finding out the characteristics of these domains is one of our future research directions.

9. Preprocessing individual action template

In the previous section, the checking algorithm requires that for every action template in Λ, all of its reductions satisfy Restriction 5.1. Sometimes not all the action templates in Λ fit this requirement, but only some of them do. In situations like this, it may still be possible to ensure that unresolvable conflicts in a plan cannot be resolved in any of the plan's reduction.

For example, suppose P is a plan which is Δ-unlinearizable, and for all the actions a in P, all the reductions of their corresponding action templates in Λ and their "descendents" satisfy Restriction 5.1. Then any composite reduction $Q(P)$ of P is $Q(\Delta)$-unlinearizable.

Before introducing the new restriction formally, we first define a notation, TC(**a**), for the *transitive closure* of action templates induced by action/subaction relationships. This notation will be used to refer to the descendents of an action template **a**. Thus, TC(**a**) represents all the action templates in Λ that can be obtained from **a** by reducing it in one or more steps using the reduction schemata in Φ. More formally, let Λ be the set of action templates and Φ be the set of action reduction schemata in a planning system's planning knowledge. Then

Definition 9.1

Let $\mathbf{a} \in \Lambda$ be an action template. TC(**a**) is defined recursively as follows:

1. $\mathbf{a} \in$ TC(**a**),
2. If $\mathbf{b} \in$ TC(**a**), then $\forall R \in \alpha(\mathbf{b})$ if $c \in A(R(\mathbf{b}))$, then template(c) $\in$ TC(**a**).

As an example of TC(**a**), consider the set of action reduction schemata in Fig. 14. For the action template **a** in the figure,

$$\text{TC}(\mathbf{a}) = \{a, a_1, a_2, a_3, a_4\}$$

For a given Λ, suppose each action template has a maximum of k action reduction schemata applicable to it. Then the relationship between actions and subactions can be represented as a directed graph, with $k \times |\Lambda|^2$ edges. For any given action template **a**, the computation of TC(**a**) can be implemented as a depth-first search in this graph. Thus, the worst-case time complexity for computing TC for one action template is $O(k \times |\Lambda|^2)$.

Now we formally present the restriction:

Restriction 9.1

Let Λ be the set of action templates and Φ be the set of action reduction schemata of a planning system's planning knowledge. Let $\mathbf{a} \in \Lambda$ be an action template. $\forall \mathbf{a}' \in$ TC(**a**) and $\forall R \in \alpha(\mathbf{a}')$, $R(\mathbf{a}')$ satisfies the Unique-Main-Subaction Restriction.

Intuitively, this restriction requires all the action templates that can be reduced to from **a** satisfy the Unique-Main-Subaction Restriction. If it is satisfied, we can prove a similar result concerning the nonexistence of solutions below any unlinearizable plan.

Theorem 9.1

Let P be a Δ-unlinearizable plan. If for every action a in the plan, template(a) satisfies Restriction 9.1, then $\text{AU}_Q(P)$ is $\text{AU}_Q(\Delta)$-unlinearizable for any composite reduction Q.

The set of action templates in Λ can be preprocessed in a way similar to the previous section. First, Algorithm *Checking* can be performed on Λ. If it returns "false," then the following processing can be done: for each action template $\mathbf{a} \in \Lambda$, if every action template in TC(**a**) is marked by Algorithm *Checking*, then **a** satisfies Restriction 9.1. Mark all the action templates in Λ which satisfies Restriction 9.1. Suppose P is a plan unresolvable with respect to its set of protection intervals. If for every non-primitive action a in P, template(a) is marked in the above sense, then no composite reduction of P is linearizable with respect to its set of protection intervals. Therefore, backtracking from P will not lose any possible solutions.

10. Point-protected conditions

In the previous sections, we have considered unresolvable conflicts among the actions in a plan, which contains a set of conditions that have to be protected throughout intervals of time. This type of condition can be referred to as *interval-protected conditions*. There is another kind of condition in planning which requires only the validity of a set of conditions right before an action is started. This type can be called *point-protected conditions*.

Point-protected conditions are more complicated than interval-protected conditions in that there are more ways to resolve an unresolvable conflict in a plan through reduction. In particular, there may be some subaction w of an action in the plan P, which achieves the precondition of some actions in a conflict, and thus resolves it. Such actions have been referred to as white knights (Chapman 1987). For instance, suppose that there is a double cross conflict involving actions a and b. A white knight in this situation could be an action w such that

1. effects(w) $\supset$ preconditions(a),
2. effects(b) $\not\supset$ $\neg$preconditions(w), and
3. effects(w) $\not\supset$ $\neg$effects(b).

Therefore the conflict can be resolved by the ordering $b \prec w \prec a$. The action w can be a subaction of some existing action c in the plan. Because w only appears in some reduction of the current plan, an unresolvable conflict in

P can be resolved in $Q(P)$. The restrictions we have developed so far do not prevent white knights from appearing, and therefore a set of new restrictions has to be developed in order for our theory to work also for point-protected conditions. This is one aspect of our future work.

11. Conclusion

We have developed a set of syntactic restrictions on the relationship between a non-primitive action and its set of subactions. These restrictions enable hierarchical planning systems to recognize a dead-end state much earlier. If these restrictions are satisfied, one can conclude that a plan is unlinearizable at any level of reduction below, given that it is unlinearizable at the current level. If this downward-unlinearizability property holds for a set of action reduction schemata, hierarchical planners need not reduce an unlinearizable plan further to explore possible solutions at some level or reduction below, and can backtrack from such a dead-end point. Moreover, because of the static nature of the restrictions, it is possible to preprocess a given set of reduction schemata to check which schema satisfies the restriction before the planning process starts.

There are a number of issues that we will address in our future work. In this paper, we have considered hierarchical planning with a particular kind of action representation, in which the preconditions and effects of an action are sets of literals. This restricts the techniques developed to be only useful for a subset of the existing hierarchical planning systems. For example, SIPE (Wilkins 1984) allows in its action representation conditions which can be context-dependent, quantified, and disjunctive. In addition, some effects can be deduced from a set of external axioms. With action representations as expressive as these, some of our results no longer apply. For example, consider actions with preconditions in the form of disjunctions of conditions. Then the effects of a group of actions can collectively deny or assert the preconditions of other actions. With these representations, the detection of interactions during planning is a more complicated problem, since at every step of planning, groups of actions have to be tested for possible interactions. We believe that the idea of extracting information about interaction a priori can be used to help ease this computational burden, in the same way this paper has shown. For example, information about which actions are more likely to interact may be obtained through preprocessing, and used to provide a better control strategy for the detection and handling of interactions.

Preprocessing can also provide a planner with good heuristics for choosing among alternative action reduction schemata. To reduce a non-primitive action, there is usually more than one available schemata, and current hierarchical planning systems select the alternatives in a depth-first manner. Depending on which reductions are chosen, different conflicts may be introduced, and this may affect the planning efficiency.

One heuristic which can be used in choosing a reduction is to minimize the amount of conflicts introduced. Preprocessing can be helpful for predicting which reductions are likely to yield a set of conflicts that is small — and just as importantly, a set of conflicts that is easy to resolve.

During hierarchical planning, a plan often has more than one non-primitive action to be reduced. A least-commitment planner delays commitment to both decisions about orderings between the actions and bindings of the variables. Therefore, a plan usually contains a set of variables yet to be bound. The order in which the non-primitive actions are reduced may affect how the variables are bound in a plan, which in turn affects both the efficiency of planning and the quality of the final plan. The order in which the non-primitive actions are reduced can be called *planning order*, which may be quite different from the temporal orderings. One of our future works is to develop techniques for obtaining a good planning order via preprocessing.

Acknowledgements

The author is indebted to Dana Nau, James Hendler, and Josh Tenenberg for many useful suggestions. The Computational Intelligence referees also provided many in-depth comments. Support for this research was provided partly by NSF Presidential Young Investigator award DCR-83-51463 to Dana Nau, and partly by an interim grant provided by the University of Waterloo.

CHAPMAN, D. 1987. Planning for conjunctive goals. Artificial Intelligence, **32**: 333–377.

CHARNIAK, E., and MCDERMOTT, D. 1984. Introduction to artificial intelligence. Addison-Wesley Publishing Co., Reading, MA. Chap. 9.

FIKES, R.E., and NILSSON, N.J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, **2**(3/4): 189–208.

KAMBHAMPATI, S. 1989. Flexible reuse and modification in hierarchical planning: a validation structure based approach. Ph.D. thesis (CS-TR-2334), University of Maryland, College Park, MD, pp. 155–157.

SACERDOTI, E.D. 1974. Planning in a hierarchy of abstraction spaces. Artificial Intelligence, **5**: 115–135.

——— 1977. A structure of plans and behavior. American Elsevier, New York, NY.

TATE, A. 1977. Generating project networks. Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, MA, pp. 888–893.

TENENBERG, J. 1988. Abstraction in planning. Technical Report TR250, Computer Science Department, University of Rochester, Rochester, NY.

WILKINS, D.E. 1984. Domain-independent planning: representation and plan generation. Artificial Intelligence, **22**: 269–301.

——— 1986. Hierarchical planning: definition and implementation. Proceedings of the Seventh European Conference on Artificial Intelligence, Brighton, United Kingdom, pp. 659–671.

——— 1988. Practical planning: extending the classical AI planning paradigm. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

YANG, Q. 1989. Improving the efficiency of planning. Ph.D. thesis, University of Maryland, College Park, MD. Available as University of Waterloo Research Report CS-89-34, Department of Computer Science, Waterloo, Ont.

Appendix A. Proofs

This appendix provides proofs for two of the main theorems, theorems 5.1 and 6.1, in the paper. The proofs to the other theorems and corollaries can be found in Yang (1989).

Theorem 5.1

Let P be a plan containing a non-primitive action a. Suppose for every other action b in P, $R(a)$ satisfies Restriction 5.1 with respect to b. Let $R(P)$ be the reduction of P

by reducing a using $R(a)$. Then if P is Δ-unlinearizable, then $R(P)$ is $\mathrm{AU}_R(\Delta)$-unlinearizable.

Below we only prove one part of Theorem 5.1, under the condition that every action in P satisfy Restriction 5.1. A similar proof can be constructed for the theorem when the actions satisfy Restriction 5.2.

Proof

Let L be a linearization of $R(P)$ that is $\mathrm{AU}_R(\Delta)$-consistent, where $R(P)$ is obtained by reducing the action a in P. Below, we provide a transformation which converts L to a linearization of P and $\mathrm{AU}_R(\Delta)$ to Δ, and show that the resultant linearization of this transformation does not violate any protection interval of Δ in L. The transformation is defined as follows:

Let the first subaction of a in L be a_1, and the last subaction of a in L be a_n. For every action $b \in A(P)$ such that $b \notin A(R(a))$ and $a_1 \prec b \prec a_k$ in L, relocate b to the left of a_1. Repeat this step for all such b starting with the leftmost one after a_1, in a left-to-right order. Also, the original ordering between the bs are still kept after the relocation, so that if $b_1 \prec b_2$ before the relocation, then $b_1 \prec b_2$ after the relocation. After all the relocations are done, merge all the main subactions of a from a_1 to a_n into a. Let the new sequence be L'.

Next, update the protection intervals in $\mathrm{AU}_R(\Delta)$ as follows:

1. If $\langle p, c, a_i \rangle \in \mathrm{AU}_R(\Delta)$ and preconditions$(a_i) \cap$ preconditions$(a) \neq \emptyset$ then replace a_i by a;
2. If $\langle p, a_j, g \rangle \in \mathrm{AU}_R(\Delta)$ and effects$(a_j) \cap$ effects$(a) \neq \emptyset$ then replace a_j by a;
3. For any other protection interval δ, if $a_i \in A(R(a))$ appear in δ, then remove δ from $\mathrm{AU}_R(\Delta)$.

Let the resultant protection intervals be Δ'. Then Δ' is Δ, and L' is a linearization of P. This transformation does not create any new conflicts, since if it does, then it must be because either some b deletes a precondition of a, or a deletes an effect of b. But this is impossible, since according to Restriction 5.1, b would have created a conflict in L before the relocation, contradicting to the assumption that L is $\mathrm{AU}_R(\Delta)$-consistent. Thus, P must be Δ-linearizable, contradicting to our original assumption. ■

The part of the theorem concerning Restriction 5.2 can be similarly proved, except the transformation is done by moving every b to the right of the last action.

To prove Theorem 6.1, first consider the following lemma:

Lemma A

Let $a \in A(P)$ be a non-primitive action satisfying Restriction 6.1. Let R be a reduction schema applicable to a. Then if P is Δ-unlinearizable, then $R(P)$ is $R(\Delta)$-unlinearizable.

Proof

Suppose $R(P)$ is $R(\Delta)$-linearizable. Let L' be a linearization of $R(P)$ which is $R(\Delta)$-consistent. From Restriction 6.1, there is a unique subaction a_m of a which asserts all the effects of a relevant for Δ, which preconditions includes all the preconditions of a relevant for Δ. Let L'' be L' with all occurrences of $a_i \neq a_m$ deleted, where $a_i \in A(R(a))$. Similarly, let Δ'' be $R(\Delta)$ with all the intervals containing $a_i \neq a_m$ removed, where $a_i \in A(R(a))$. Then L'' is Δ''-consistent, since the above operations did not create any conflicts.

Now replace every occurrence of a_m in both L'' and Δ'' by a. Δ'' will be transformed to Δ, and the resultant sequence L must be Δ-consistent. This is because all of a_m's effects relevant to Δ are kept after this replacement, and any effects in effects(a)–effects(a_m) are irrelevant and not harmful to Δ. However, L is a linearization of P. Therefore, P must be Δ-linearizable, contradicting to the assumption of the lemma. ■

Theorem 6.1

Every set of action reduction schemata satisfying Restriction 6.2 is downward-unlinearizable.

Proof

Let Φ be a set of action reduction schemata, and let P be a plan which is Δ-unlinearizable. We prove the theorem by induction on the number of reductions applied to P. Specifically, we prove that for any $i \geq 0$, $Q(P)$ is $Q(\Delta)$-unlinearizable, where $Q(P)$ is any composite reduction of P produced by reducing i non-primitive actions, using the reduction schemata in Φ.

The base case, $i = 0$, corresponds to the plan P without any reduction. Since P is assumed Δ-unlinearizable, the theorem holds for the base case.

For the inductive hypothesis, assume that $Q(P)$ is $Q(\Delta)$-unlinearizable, where $Q(P)$ is a composite reduction of P produced by reducing $n \geq 0$ actions.

Let $Q'(P)$ be produced by reducing non-primitive action in $Q(P)$, and let the corresponding reduction of $Q(\Delta)$ be $Q'(\Delta)$. Since every action in $Q(P)$ is an instance of some action in Λ and since Φ satisfies Restriction 6.2, from Lemma A any reduction $Q'(P)$ of $Q(P)$ must be $Q'(\Delta)$-unlinearizable. Thus, the theorem holds for any composite reduction of P. ■

Appendix B. Action templates and reduction schemata for the blocks-world domain

This appendix provides a representation of the planning knowledge in the blocks-world domain.[2] In this domain, there are a table and a number of equally sized blocks. A block can have at most one other block immediately on top of it. A robot can stack blocks on top of each other, or put a block on the table. The table is considered to be always clear. The robot can handle only one block at a time.

The following actions are non-primitive:

1. achieve(On(x, y))
   comment: a goal for achieving On(x, y).
   preconditions = {}
   effects = {On(x, y)}
2. achieve(Ontable(x))
   comment: a goal for achieving Ontable(x).
   preconditions = {}
   effects = {Ontable(x)}
3. achieve(Cleartop(x))
   comment: a goal for achieving Cleartop(x).
   preconditions = {}
   effects = {Cleartop(x)}

[2]This domain representation is adopted from Kambhampati (1989).

4. **makeon-block-1**(x, y)
   comment: put block x on the top of block y.
   preconditions = {Block(x), Block(y), Cleartop(x), Cleartop(y)}
   effects = {$\neg$ Cleartop(y), On(x, y)}
5. **makeon-table-1**(x)
   comment: move x to the table
   preconditions = {Block(x), Cleartop(x)}
   effects = {Ontable(x)}
6. **makeon-block-2**(x, y, z)
   comment: move x from the top of y to the top of z.
   preconditions = {Block(x), Block(y), Block(z), On(x, y), Cleartop(z), Cleartop(x) $x \neq y$, $x \neq z$, $y \neq z$}
   effects = {Cleartop(y), On(x, z) $\neg$ Cleartop(z)}
7. **makeon-table-2**(x, y)
   comment: move x from the top of y to the table.
   preconditions = {Block(x), Block(y), Cleartop(x), On(x, y)}
   effects = {Ontable(x), Cleartop(y)}

The following actions are primitive:

1. **put-block-on-block**(x, y, z)
   comment: move x from the top of y to the top of z.
   preconditions = {Block(x), Block(y), Block(z), On(x, y), Cleartop(z), Cleartop(x) $x \neq y$, x, $\neq z$, $y \neq z$}
   effects = {Cleartop(y), On(x, z), $\neg$ Cleartop(z), $\neg$ On(x, y)}
2. **put-block-on-table**(x, y)
   comment: move x from the top of y to the table.
   preconditions = {Block(x), Block(y), Cleartop(x), On(x, y)}
   effects = {Ontable(x), Cleartop(y), $\neg$ On(x, y)}

Below are action reduction schemata:

1. R_1(achieve(On(x, y))):
   actions: {achieve(Cleartop(x)), achieve(Cleartop(y)), makeon-block-1(x, y)}
   orderings: {achieve(Cleartop(x)) $\prec$ make-on-block-1(x, y) achieve (Cleartop(y)) $\prec$ make-on-block-1(x, y)}
   protection intervals: {⟨Cleartop(x), achieve(Cleartop(x)), makeon-block-1(x, y)⟩ ⟨Cleartop(y), achieve(Cleartop(y)), makeon-block-1(x, y)⟩}
2. R_2(achieve(Cleartop(x))):
   actions: {achieve(Cleartop(y)), makeon-block-2(y, x, z)}
   orderings: {achieve(Cleartop(y)) $\prec$ makeon-block-2(y, x, z)}
   protection intervals: {⟨Cleartop(y), achieve(Cleartop(y)), makeon-block-2(y, x, z)⟩}
3. R_3(achieve(Cleartop(x))):
   actions: {achieve(Cleartop(y)), makeon-table-2(y, x)}
   orderings: {achieve(Cleartop(y)) $\prec$ makeon-table-2(y, x)}
   protection intervals: {⟨Cleartop(y), achieve(Cleartop(y)), makeon-table-2(y, x)⟩}
4. R_4(achieve(Ontable(x))):
   actions: {achieve(Cleartop(x)), makeon-table-1(x)}
   orderings: {achieve(Cleartop(x)) $\prec$ makeon-table-1(x)}
   protection intervals: {⟨Cleartop(x), achieve(Cleartop(x)), makeon-table-1(x)⟩}
5. R_5(**makeon-block-1**(x, y)):
   actions: {put-block-on-block(x, z, y)}
   orderings: {}
   protection intervals: {}
6. R_6(**makeon-block-2**(x, y)):
   actions: {put-block-on-block(x, z, y)}
   orderings: {}
   protection intervals: {}
7. R_7(**makeon-table-1**(x)):
   actions: {put-block-on-table(x, y)}
   orderings: {}
   protection intervals: {}
8. R_8(**makeon-table-2**(x, y)):
   actions: {put-block-on-table(x, y)}
   orderings: {}
   protection intervals: {}

In addition to the above reduction schemata, every non-primitive action of the form "achieve($C(x)$)" can also be reduced to "no-op," which is a primitive action requiring no real action. no-op corresponds to a phantom goal in NONLIN (Tate 1977) or SIPE (Wilkins 1984), and can be considered as a primitive action whose precondition and effect are both the condition $C(x)$ to be achieved.