Title: **Flexible Reuse of Plans via Annotation and Verification**

Contact: Subbarao Kambhampati
James A. Hendler

Center for Automation Research
Department of Computer Science
University of Maryland
College Park, MD 20742
(*email: rao@alv.umd.edu*)

Topic: Principles: Plan Reuse, Adaptation of Plans

Abstract: We present an approach for flexible reuse of old plans in the presence of a generative planner. In our approach the planner leaves information relevant to the reuse process in the form of annotations on every generated plan. To reuse an old plan in solving a new problem, the old plan along with its annotations is mapped into the new problem. A process of *annotation verification* is used to locate applicability failures and suggest refitting tasks. The planner is then called upon to carry out the suggested modifications—to produce an executable plan for the new problem. This integrated approach obviates the need for any extra domain knowledge (other than that already known to the planner) during reuse and thus affords a relatively domain independent framework for plan reuse. We describe the details of annotation and plan reuse in PRIAR, a plan reuse system based on this approach. We believe that our approach for plan reuse can be profitably employed by generative planners in many applied domains.

Status: Research

Effort: 1.5 person-years

The support of the Defense Advanced Research Projects Agency and the U.S. Army Center for Night Vision and Electro-Optics under Contract DAAB07-86-K-F073 is gratefully acknowledged.

CH2712-8/89/0000/0037$01.00 © 1989 IEEE

1. Introduction

The value of enabling a planning system to remember the plans it generates for later use was acknowledged early in planning research [9]. Many early systems, starting with STRIPS, generated macro-operators to store plans and reuse them. Unfortunately, macro-operators built from sequences of primitive steps proved to be inflexible for reuse because they could only be used in a limited number of situations where some sub-sequence of the old plan is applicable. A reason for this inflexibility is that macro-operators do not represent intermediate decisions and dependencies corresponding to their internal steps. More recently, the issues of reuse have been addressed primarily in case-based systems[2, 11]. However, much of the work in case-based planning has concentrated on the indexing and retrieval aspects of the old plans, rather than on the reuse methodology. Other work, focusing on reuse of designs via design replay [13, 20], aims to ease the burden of redesign rather than automating it. These design-replay systems have not addressed the issues of involved in focusing and automating adaptation.

Our current research aims at providing an integrated framework for flexible reuse of plans. This involves localization of applicability failures, characterization of these failures , suggestion of appropriate refit tasks, and finally control of the refitting. The main theme of our research is that the internal dependency structure of the plan, annotated by a planner during the planning, can be utilized in tackling these issues. In our reuse framework, the planner leaves information relevant to the reuse process in the form of annotations on every generated plan. To reuse an old plan in solving a new problem, the old plan, along with its annotations, is mapped onto the new problem. A process of ***annotation verification*** is used to locate applicability failures and suggest refitting tasks. These refit tasks suggest appropriate modifications to the plan in such a way that the applicable portions of the plan are left unaffected as much as possible, thus localizing the refitting. The planner is then called upon to carry out the suggested modifications, and produce an executable plan for the new problem. This neatly separates the adaptation decisions (eg. whether to replace a failing step or establish its preconditions, where to place a refit task, how to control the refitting) from the planning decisions (eg. how to find a replacement, how to ensure that the plan steps will not interact), and leaves the planning decisions to the generative planner.

In the rest of the paper, we describe in detail the framework we have developed for reusing the plans generated by a domain independent nonlinear planner. Our emphasis in the present paper is on the flexible reuse methodology rather than on the retrieval issues. PRIAR is a prototype system based on this framework that we have implemented for blocks world. Examples from this system are used throughout the discussion. We are investigating application of these techniques to applied domains such as process planning in automated manufacturing [4], where automated reuse is very valuable. (See [14] for a preliminary discussion.)

The terminology used in this paper is generally consistent with the previous descriptions of plan reuse [2, 11]. In particular, we use the term ***adaptation*** to denote the process of transforming a retrieved plan into a solution for the new problem. *Adaptation* involves locating applicability failures and modifying them. *Refitting* denotes the process of modifying these failures to produce a consistent plan for the new problem. For reasons that will become clearer later, we make a distinction between planneable and unplanneable conditions for operator applicability[1]. In accordance with STRIPS terminology, we call the former *preconditions* and the latter *filter conditions*. The term *applicability condition* is used to denote either filter condition or precondition. The terms *step* and *operator* are used interchangeably.

2. Reusing plans generated by a Nonlinear Planner: Overview

PRIAR system implements plan reuse capability for a domain independent, nonlinear, conjunctive planner. The planner in the system, based on NONLIN [22], has the capability of reducing a task network into an executable plan, and resolving interactions between steps. Unlike normal nonlinear planners, however, it also annotates the plans it generates and stores each annotated plan in its plan library.

Given a new planning problem consisting of an input situation and a goal specification, the reuse procedure progresses in the following stages:

1. Retrieval: A plan for a similar problem is retrieved from the planner's library. (If the retrieval fails to find a similar plan, the planner is invoked to solve the problem generatively.)

2. Interpretation: The old plan along with its annotations is interpreted in the new problem situation.

3. Annotation Verification: The annotations of the interpreted plan are verified and upon finding various forms of verification failure, various types of refit tasks are suggested.

4. Construction and reduction of reuse schema: A schema containing applicable portions of the interpreted old plan is constructed. The refit tasks suggested by the annotation verification are incorporated into this schema. The planner is then invoked to reduce this reuse schema, generating a plan for the new problem.

In the following sections, we describe the annotation framework and the reuse procedure in detail. Throughout this discussion, we will be following an example in which PRIAR reuses a plan for the three block stacking problem shown in Figure 1 in finding a plan for the four block stacking problem depicted in Figure 2.

3. Annotation Framework

Annotations on a plan step reflect the dependencies between that plan step and the rest of the plan. An annotated plan produced by PRIAR explicitly represents the validation states between successive steps of the plan. A state following a plan step consists of the useful outcomes of that plan step, and those facts of the previous state that have to persist over this plan step for the validation of the plan. The former are linked to the plan step by **o** links and the latter are linked to the corresponding facts in the previous state by **p** links[2]. The initial state consists of the facts of the input situation that are necessary for plan validation and the final state consists of the goals achieved by the plan. All the applicability conditions (eg.

[1] This type of distinction has been made under various names previously in the planning literature (see [1, 22]).

[2] The states of the annotated plan do not bear relation to the states of linear planners. The planner still maintains a distributed world model during planning. These states have structural similarity to *kernels* of triangle tables [9] (see section 9).

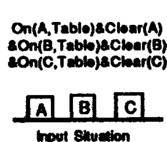

Figure 1. Three block stacking problem

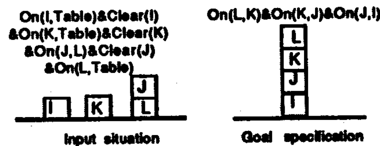

Figure 2. Four block stacking problem

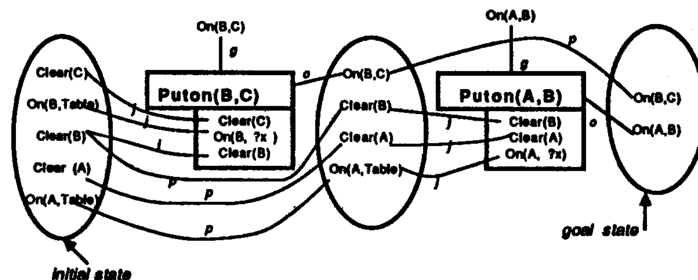

Figure 3. Annotated Plan for three block stacking problem

preconditions, filter conditions) of a plan step are connected to their justifiers in the immediately preceding state by **j** links. The goals achieved by each plan step are connected to the step by **g** links. These may either be the goals of the overall plan or the subgoals set up by the task reduction.

The links **p**, **j**, and **o** are called validation links. In the annotations of an executable plan, all goals in the final state should be supported by validation chains (facts or conditions connected by validation links), and all validation chains should be grounded in the initial state. Each state of an annotated plan contains only those facts that are required for the validation of the rest of the plan.

Figure 3 graphically depicts the annotated plan generated by the planner for the three block stacking problem shown in figure 1. Consider the validation of the goal $On(B,C)$. $On(B,C)$ is true in the *goal-state* because it is supported by a **p** link from the fact $On(B,C)$ in *state*-1. $On(B,C)$ in *state*-1 is supported by an **o** link from *step*-1. *step*-1 in turn is valid because all its applicability conditions are supported by **j** links from the *initial-state*. Thus, the goal $On(B,C)$ is validated. The validation of any given fact or condition can be verified similarly by following the validation links.

In PRIAR, these annotations are derived easily from the incremental records kept by the planner during plan generation in the datastructures TOME and GOST. These are datastructures used by PRIAR's NONLIN based planner (see [22] for details). Whenever the planner expands a task, it records the *effects* of the expansion in TOME. The *applicability conditions* of the expansion and their contributors are recorded in GOST. These (or similar) datastructures are used by most nonlinear planners to aid in plan generation. Thus annotating a plan does not significantly increase the cost of planning. However, deriving these annotations after the planning, without the aid of the incremental records kept by the planner, can be a very costly operation. Thus it is advantageous to integrate planning and plan reuse.

4. Retrieval and Mapping

The purpose of the retrieval is to find a plan from the planner's library that can be efficiently adapted to a given problem. For the blocks world, PRIAR currently uses a simple goal based retrieval method, that selects the plan matching the most goals of the new problem. The matching is done by a modified unification algorithm which can partially unify formulas with differing number of conjuncts (see [15] for details). If a unique mapping is not produced by this process, the system currently selects one of the mappings arbitrarily. Since the rest of the reuse procedure is capable of adapting the plan with any of the remaining mappings, this will not undermine the reuse capability.

In the current example, we will assume that the three block stacking problem matches the most goals. The partial unification gives two mappings $[A \to L, B \to K, C \to J]$ and $[A \to K, B \to J, C \to I]$. The former is selected as the object mapping arbitrarily.

The current retrieval procedure is not claimed to be the best retrieval procedure for the blocks world. As mentioned earlier, we concentrate on the reuse methodology rather than on the retrieval in this research. However, it will become clear that in this framework best match retrieval is not critical for successful reuse. This facilitates the use of much cheaper syntactic retrieval methods in many applied domains. For example, in process-planning for automated manufacturing domain, group-technology coding based retrieval methods are quite effective [4, *chapter 4*].

5. Interpretation

The purpose of interpretation is to map the retrieved plan along with its annotations into the new problem, taking the differences between the old and new situations into account. The interpretation step consists of mainly syntactic operations. It serves to focus the annotation verification procedure towards those parts of the retrieved plan that have potentially failing validations.

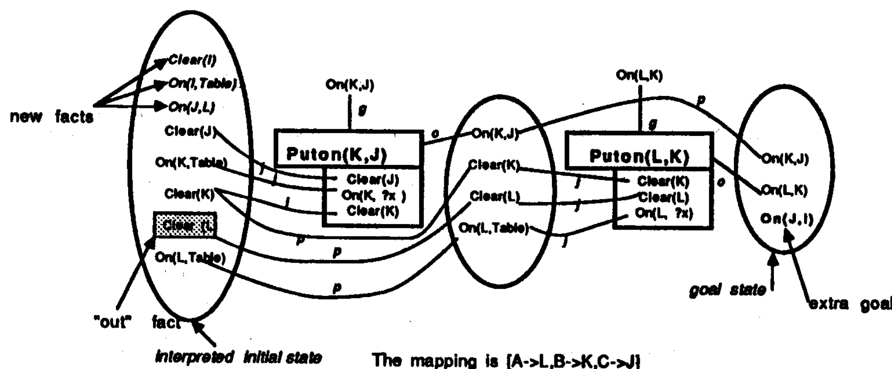

Figure 4. Three block stacking plan interpreted for Four block stacking problem

To do this, first the object mapping supplied by the retrieval procedure is used to map the states, goals and the applicability conditions of the retrieved annotated plan into the new problem. Next, the facts in the initial state of the mapped plan that are not true in the input situation of the new problem are marked *out*. The facts which are true in the new input situation, but which are not present in the initial state, are added to the initial state and are marked *new*. A similar processing is done to the final state of the mapped old plan; marking the goals not required in the new problem *unnecessary*, and adding any additional goals of the new problem to the final state as *extra* goals. After this, if any goals in the final state of the interpreted plan are also present in the initial state (without an *out* mark), they are marked *p-phantom*, to signify that they are potentially phantom goals[3]. The steps achieving such goals in the old plan may be redundant in the new problem situation. The annotation links of the mapped plan remain unaffected. At the end of this processing, we will have a partial plan that is consistent with the old and new problem specifications.

Figure 4 shows the three block stacking plan interpreted in the four block stacking problem situation. *Clear(L)* is no longer true in the input specification of the new problem. So it is marked *out* in the initial state of the interpreted plan. The facts *On (J,L)*, *On (I,Table)* and *Clear (I)* are true in the new problem but were not true in the old problem, so they are added to the initial state. Similarly, *On(J,I)* is an extra goal and is added to the final state. There are no *unnecessary* goals.

6. Annotation Verification

The purpose of this procedure is to suggest modifications to be made to the interpreted plan to transform it into a solution for the new planning problem. In the current system, annotation verification suggests three classes of modifications—removing steps that achieve unnecessary goals, adding tasks to achieve any extra goals and suggesting refit tasks for failing validations. Figure 5 outlines the annotation verification procedure.

[3] Phantom goals refer to the goals of a planning problem that are achieved without step addition [21]. Such goals get established either through helpful interactions from other steps in the plan, or by persistence from the initial state.

```
Procedure Annot_verify (Iplan : Interpreted Plan)
{
  foreach goal ∈ Goal_state(Iplan ) do {
    if goal is marked unnecessary
      then remove validations terminating in goal
    if goal is marked extra
      then add task for the extra goal } od
  foreach out marked fact ∈ Initial_state(Iplan ) do {
    foreach validation affected by fact do {
      if the validation is to a goal
        then dephantomize the goal (see below)
      if the validation is to an applicability condition
        then if it is a failing filter condition
                then suggest a replacement task
             if it is a failing precondition
                then suggest a precondition establishment task }} od
}
```

Figure 5. Annotation Verification Procedure

Removing unnecessary steps: The plan steps that achieve *unnecessary* goals of the final state can be removed from the plan, along with any steps whose sole purpose is supplying applicability conditions for such plan steps. Such a modification is necessitated if the retrieved plan is more complex than required by the new problem. The removal of a plan step is effected by merging the states immediately preceding and following it, and removing any validation chains emanating from or terminating in it. After removing all the unnecessary steps, the corresponding goals are also removed from the final state. The result will be a properly annotated plan without the unnecessary steps and goals.

Removal of steps achieving *p-phantom* goals (goals already true in the initial state) is postponed until the reuse schema is reduced by the planner. This is done because the *p-phantom* goals may not remain phantom after the reduction.

Adding tasks for extra goals: Any goal in the goal state of the interpreted plan that is not supported by a validation link is an extra goal. The extra goals are treated as new conjuncts for PRIAR's nonlinear planner. No attempt is made to order the task achieving extra goal with respect to the existing tasks. Instead, for each extra goal, G, a task of the form *Achieve* $[G]$, with an outcome G, is created and is added in parallel to the interpreted plan (i.e., between initial and final states). This process leaves the onus of reducing the goal and producing an interactionless plan on the planner, where it rightfully belongs.

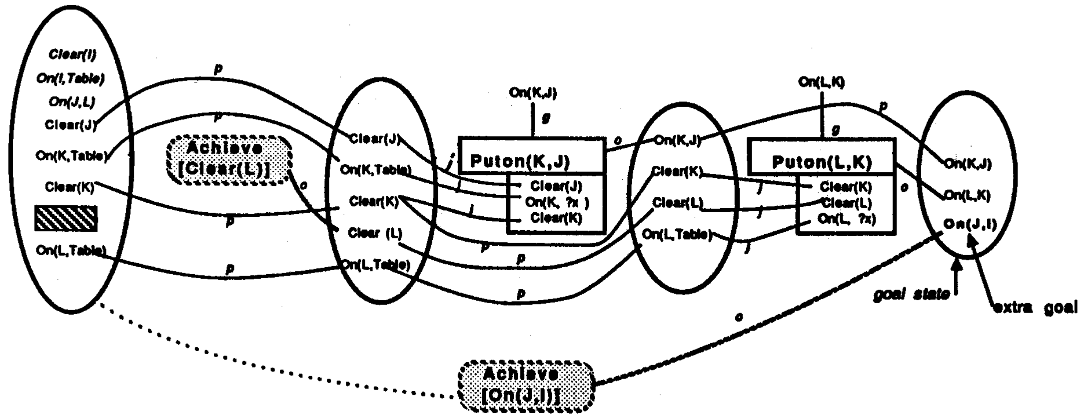

Figure 6. Task network constructed by the annotation verification procedure for the four block stacking problem

Locating and modifying failing validations: The *out* facts of the initial state of the interpreted plan may be supplying validation chains to the applicability conditions of plan steps or goals of the old plan. Such conditions (called failing conditions), and goals (called failing phantom goals), can be easily located by following validation links emanating from the *out* facts. The treatment of a failing conditions depends upon the type of the condition. We consider two types of conditions—filter conditions and preconditions.

1. Refit tasks suggested for Failing preconditions: For every failing precondition, P, a task *Achieve* $[P]$ should be added to the interpreted plan before the plan step requiring that precondition. The obviously correct strategy would be to add a task to achieve the failing condition in parallel between the initial state and the state preceding the step that requires the precondition. However, in PRIAR, a general heuristic followed is to establish a failing precondition at the place where its validation chain fails. The rationale is that since the plan being reused was itself a valid plan, the place from where it established the failing condition is a good place for adding the new task to re-establish the condition. This choice would possibly reduce subgoal interactions. Following this heuristic, the task *Achieve* $[P]$ is added immediately after the initial state. A new state is introduced immediately after the added task. Annotations are adjusted so that the new task provides a validation chain to the failing precondition.

2. Refit tasks suggested for Failing filter conditions: If the validation for a filter condition of a plan step S fails because of an *out* fact in the initial state, then the step S should be replaced. This is done by finding the goals G_S, of the step S, and replacing S by a task to achieve G_S conjunctively. The new task will now be supplying the validation chains previously supplied by S. Control information can be added to the new task specifying that S should not be tried again to achieve G_S[4]. Any steps that supply validation chains to the conditions of the replaced step S are potentially redundant, and should be removed unless they are also supplying validations to other parts of the plan.

[4] This brings up an interesting facet of annotation verification—the ability to specify control information to the planner. We will have more to say about this in section 10.

3. Refit tasks suggested for Failing phantom goals: The phantom goals whose validations are failing should be dephantomized. This involves adding tasks to the interpreted plan to achieve these goals. In contrast to the extra goals, the tasks to achieve de-phantomized goals can be placed right after the contributor state (the state that made this goal phantom). The rationale for this is that in the old plan the validation for these goals persisted over all the subsequent steps of the plan.

Example. Figure 6 shows the task network produced by the annotation verification procedure for the four block stacking problem. The input to the annotation verification procedure is the interpreted plan shown in Figure 4. This example contains a failing precondition and an extra goal. The goal $On(J,I)$ of the final state of the interpreted plan is an extra goal. So, the refit task *Achieve* $[On(J,I)]$ is added to the task network, in parallel to the existing plan. It now supplies validation chain to $On(J,I)$ in the final state. Next, the fact $Clear(L)$, which is *out* in the initial state, causes the validation chain of the precondition $Clear(L)$, of the step $Puton(L,K)$, to fail. So, the refit task *Achieve* $[Clear(L)]$ is added at the initial state. A new state is constructed after *Achieve* $[Clear(L)]$ and the annotations are updated appropriately. *Achieve* $[Clear(L)]$ now provides the validation chain for the failing precondition.

7. Construction and Reduction of reuse schema

After annotation verification, PRIAR converts the annotated plan with the suggested refit tasks into a reuse schema that its planner understands. This resue schema representation has the same syntax as other goal reduction schemas of PRIAR's planner. Since the annotated plan contains all the information about the inter-step dependencies, the reuse schema can start the planner off correctly. Figure 7 shows PRIAR's representation of the reuse schema constructed for the four block stacking example. The correspondence between the reuse schema and the annotated plan structure in Figure 6 should be obvious.

Next the planner is asked to solve the four block stacking problem using the reuse schema supplied. The planner can easily derive all the information necessary to start planning on the partially reduced task network represented by the resue schema. PRIAR's planner reconstructs GOST and TOME struc-

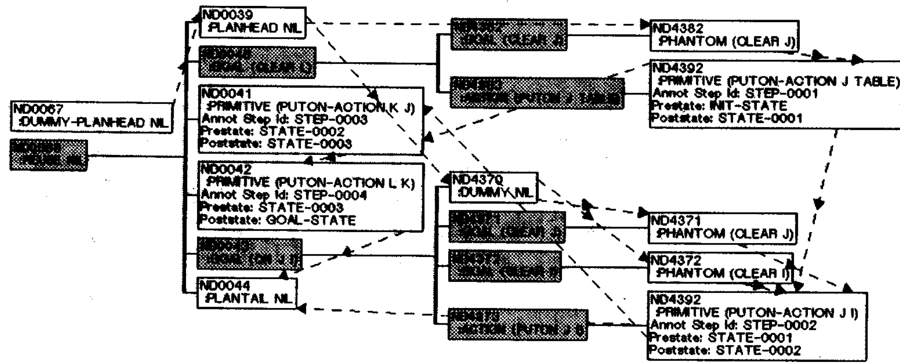

Figure 8. Hierarchical structure of the Plan for four block stacking problem produced by Reducing the reuse schema

```
the reuse schema constructed is
  (LAMBDA ()
    (STORE-ALWAYS-CTXT '((CLEAR TABLE)))
    (STORE-INIT-CTXT '((ON J L) (CLEAR I) (ON I TABLE)))
    (REUSE :EXPANSION
        ((BEG :PLANHEAD) (NEWSTEP-8 :GOAL (CLEAR L))
         (STEP-1 :PRIMITIVE (PUTON-ACTION K J))
         (STEP-2 :PRIMITIVE (PUTON-ACTION L K))
         (NEWSTEP-2 :GOAL (ON J I)) (END :PLANTAIL))
        :ORDERINGS
        ((NEWSTEP-8 -> STEP-1) (STEP-1 -> STEP-2) (STEP-2 -> END)
         (NEWSTEP-2 -> END) (BEG -> NEWSTEP-2) (BEG -> NEWSTEP-8))
        :CONDITIONS
        ((:USE-WHEN (ON K TABLE) :AT STEP-1 :FROM BEG)
         (:PRECOND (CLEAR J) :AT STEP-1 :FROM BEG)
         (:PRECOND (CLEAR K) :AT STEP-1 :FROM BEG)
         (:USE-WHEN (ON L TABLE) :AT STEP-2 :FROM BEG)
         (:PRECOND (CLEAR L) :AT STEP-2 :FROM NEWSTEP-8)
         (:PRECOND (CLEAR K) :AT STEP-2 :FROM BEG)
         (:PRECOND (ON J I) :AT END :FROM NEWSTEP-2)
         (:PRECOND (ON K J) :AT END :FROM STEP-1)
         (:PRECOND (ON L K) :AT END :FROM STEP-2))
        :EFFECTS
        ((BEG :ASSERT (CLEAR K)) (BEG :ASSERT (CLEAR J))
         (BEG :ASSERT (ON K TABLE)) (BEG :ASSERT (ON L TABLE))
         (NEWSTEP-8 :ASSERT (CLEAR L))
         (STEP-1 :DELETE (ON K TABLE)) (STEP-1 :DELETE (CLEAR J))
         (STEP-1 :ASSERT (ON K J)) (STEP-2 :DELETE (ON L TABLE))
         (STEP-2 :DELETE (CLEAR K)) (STEP-2 :ASSERT (ON L K))
         (NEWSTEP-2 :ASSERT (ON J I)))))
```

Figure 7. Reuse schema for four block stacking problem

tures [22] from the reuse schema and proceeds to reduce any unreduced tasks in the task network. This process of reduction of the reuse schema is not different from generative planning, except that the planner is given an already partially reduced task network to start with.

After the reuse schema is reduced, the planner should check for any redundant steps in the plan. Such steps may be achieving *p-phantom* goals (see section 5), or may have been supplying preconditions to a replaced step. This check can be done efficiently while re-annotating the plan—the steps that do not supply validation chains to any goal achieving steps in the plan are redundant and should be removed. Figure 8 shows the hierarchical structure of the plan for the four block stacking problem produced by reducing the reuse schema in Figure 7. The dashed arrow lines show the precedence relations between the steps of the plan developed. The figure shows clearly that the planner starts with a partially instantiated plan and reduces only the unachieved goals (corresponding to the refit tasks suggested by the annotation verification procedure), to efficiently generate the plan for the four block stacking problem.

8. Discussion

The annotation verification procedure in our approach preserves the applicable portions of the old plan, and leaves the inter-step order of the old plan undisturbed to the extent possible. In some cases, it even places the refit-tasks in such a way as to reduce interactions with the rest of the plan. Because of this many goals need not be re-achieved and many interactions do not have to be re-analyzed during reduction of the reuse schema. For example, in the reuse schema of Figure 7, since the steps *Puton* (K,J) and *Puton* (L,K) are left in the same order as in the old plan, they would not interact with each other. It can be seen from Figure 8 that they are left untouched in the final plan. Similarly, since *Achieve* [*Clear* (L)] is added at the initial state, it does not cause any interactions with other steps either. Since even the simplest type of domain-independent nonlinear planning is exponential [5], these capabilities can lead to significant reduction in the overall complexity of planning. Further, the reuse procedure exhibits flexibility by being able to partially reuse any applicable portions of the retrieved plan. Such capabilities of the reuse procedure are a direct result of storage and utilization of internal dependency structure of the plans to locate and characterize applicability failures.

The process of annotation does not increase the cost of generative planning significantly. In many cases, such annotations are useful for even a stand-alone generative planner for the purposes of plan revision and execution monitoring (see next section). The reuse approach presented is relatively domain and planner independent. It does not use any domain knowledge other than that explicitly represented in the annotations.

The worst case complexity of reuse of a plan based on this approach will not be significantly expensive compared to planning from scratch. In the worst case, if the retrieved plan is totally inapplicable for reuse, the reuse procedure will degenerate into generative planning. The complexity of annotation verification, which is the main part of the reuse procedure, is polynomial in the number of affected validations. The efficiency of the retrieval methods is still important. Much of the complexity of retrieval methods is due to insistence on best match retrieval. However, given the flexibility of our reuse approach, retrieval of the best match plan is no longer critical.

9. Related Work

Alterman's adaptive planning system, PLEX [2] relies on the place of a plan in the background of other plans in the plan library to guide the adaptation. It starts with a highly structured plan library. CHEF[11], a case-based planner, stores its

plans indexed by the goals they achieve and the interactions they avoid. CHEF's stored plans do not have justification structure; they are adapted interpretively by a test-debug strategy. In contrast to PLEX and CHEF, our approach couples the information relevant to adaptation along with the individual plans. In this sense, it is similar to Carbonell's [3] proposed methodology for problem solving by analogy by remembering a full derivational history along with every problem solution. The JULIA system [6, 17] remembers derivational information explicitly in datastructures called *value frames*. However, according to Kolodner [18], it uses such information only to check the applicability of the retrieved solution to the new problem situation. Other related work includes Tenenberg's [23] proposal of structures called *plan graphs* (which can be seen as generalized triangle tables) to focus adaptation.

Plan internal dependency structure representations similar to our annotation scheme have been used by several planners previously to localize failures in replanning (repairing a plan in response to an execution failure). Examples include triangle tables of [8], subgoal and decision graphs of [12], and more recently a proposal to maintain plan decision dependencies with the help of ATMS [10]). However, such work does not concentrate on characterizing the failures and specifying refitting tasks. Though the contents of PRIAR's annotations are some what similar to the contents of the triangle tables [9], there are important functional differences. PRIAR's annotations afford flexibility in reuse by allowing partial reuse of retrieved plans, and help in localizing and characterizing applicability failures. In contrast, triangle tables only allow reuse of either an entire macro-operator or an explicit subsequence of it.

10. Conclusion

In this paper, we have presented an integrated approach to plan reuse that utilizes the internal dependency structure of the stored plans to yield a flexible reuse framework without recourse to additional domain models. We have shown that a careful characterization of the applicability failures allows the reuse procedure to suggest appropriate refitting tasks that lead to efficient refitting. There are, however, some important limitations to this framework. While the current framework can suggest appropriate refit tasks using the dependency information, it cannot control the planner during the reduction of these refit tasks. This latter ability is very important in most domains where the planner's search space has high branching factor resulting in many choices. We are currently investigating ways of controlling refitting. An important criterion for the efficiency of refitting is that it should leave the applicable portions of the plan unaffected. We found that a powerful strategy which enforces this is to prefer the operators/steps that preserve the conditions that persisted over the replaced step (via **p** links) as replacement choices. This strategy can be further refined by giving priorities to such conditions and preferring the operator that preserves the greatest number of higher priority conditions. We have also found [15] that extending the current annotation framework to capture plan decision dependencies [7, 12] can help in controlling the refitting. In [16] we describe a refitting control strategy based on these ideas for PRIAR. Other systems that address refitting control issues include Mittal et al's PRIDE [19] and Turner et al's CAS [24].

References

1. J. F. Allen and C. R. Perrault, "Analyzing Intention in Utterances", *Artificial Intelligence 15* (1980), 143-178.
2. R. Alterman, "An Adaptive Planner", *Proceedings of AAAI*, 1986, 65-69.
3. J. G. Carbonell, "Derivational Analogy and its Role in Problem Solving", *Proceedings of AAAI*, Washington D.C., 1983, 64-69.
4. T. Chang and R. A. Wysk, *An Introduction to Automated Process Planning Systems*, Prentice Hall, Englewood Cliffs, NJ, 1985.
5. D. Chapman, "Planning for Conjunctive Goals", *Artificial Intelligence 32* (1987), 333-377.
6. R. E. Cullingford and J. L. Kolodner, "Interactive Advice Giving", *Proceedings of the IEEE International Conference on Systems, Man, Cybernetics*, 1986, 709-714.
7. L. Daniel, "Planning: Modifying non-linear plans", DAI Working paper 24, University of Edinburgh, December 1977.
8. R. Fikes and N. Nilsson, "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence 2* (1971), 189-208.
9. R. Fikes, P. Hart and N. Nilsson, "Learning and Executing Generalized Robot Plans", *Artificial Intelligence 3* (1972), 251-288.
10. R. Fikes, "Use of Truth Maintenance in Automatic Planning", *Proceedings of Darpa Workshop on Knowledge-Based Planning*, 1987.
11. K. J. Hammond, "CHEF: A Model of Case-Based Planning", *Proceedings of AAAI*, 1986, 267-271.
12. P. J. Hayes, "A Representation for Robot Plans", *Proceedings of IJCAI*, Tbisili, U.S.S.R, 1975.
13. M. N. Huhns and R. D. Acosta, "ARGO: An analogical reasoning system for solving design problems", Technical Report AI/CAD-092-87, Microelectronics and Computer Technology Corporation, March 1987.
14. S. Kambhampati and J. A. Hendler, "Adaptation of Plans via Annotation and Verification", *First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Tullahoma, TN, 1988.
15. S. Kambhampati, "An Approach for Flexible Reuse of Plans", CS-Tech. Rep.-2054 and CAR-Tech. Rep.-367, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD 20742, June 1988.
16. S. Kambhampati and J. A. Hendler, Control of Refitting during Plan Reuse, (Submitted to IJCAI-89), December 1988.
17. J. Kolodner, "Some Little-Known Complexities of Case-Based Inference", *Proceedings of TICIP*, 1986. " in "Case-Based Inference: A collection of Papers," GIT-ICS-87.
18. J. Kolodner, "Case-Based Problem Solving", *Proceedings of the Fourth International Workshop on Machine Learning*, University of California, Irvine, June 1987.
19. S. Mittal and A. Araya, "A Knowledge-Based Framework for Design", *Proceedings of Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, 856-865.
20. J. Mostow and M. Barley, "Automated Reuse of Design Plans", *Proceedings of International Conference on Engineering Design*, 1987.
21. E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier North-Holland, New York, 1977.
22. A. Tate, "Generating Project Networks", *Proceedings of 5th IJCAI*, 1977, 888-893.
23. J. Tenenberg, "Planning With Abstraction", *Proceedings of AAAI*, August 1986, 76-80.
24. R. M. Turner, "Issues in the Design of Advisory Systems: The Consumer-Advisor System", GIT-ICS-87/19, School of Information and Computer Science, Georgia Institute of Technology, April 1987.