Hybrid Planning for Partially Hierarchical Domains

Subbarao Kambhampati*, Amol Mali & Biplav Srivastava
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
email: {rao,amol.mali,biplav}@asu.edu
WWW: http://rakaposhi.eas.asu.edu/yochan.html

Abstract

Hierarchical task network and action-based planning approaches have traditionally been studied separately. In many domains, human expertise in the form of hierarchical reduction schemas exists, but is incomplete. In such domains, hybrid approaches that use both HTN and action-based planning techniques are needed. In this paper, we extend our previous work on refinement planning to include hierarchical planning. Specifically, we provide a generalized plan-space refinement that is capable of handling non-primitive actions. The generalization provides a principled way of handling partially hierarchical domains, while preserving systematicity, and respecting the user-intent inherent in the reduction schemas. Our general account also puts into perspective the many surface differences between the HTN and action-based planners, and could support the transfer of progress between HTN and action-based planning approaches.

1 Introduction

Traditionally, classical planning problem is posed as one of finding an action sequence that will take an agent from a specified initial state of the world to a desired goal state, given only the description of the executable actions in the domain. In many realistic domains, there exist human experts, who are ready to share their significant planning experience with automated planners. Efficiency of a planner, as well as the acceptability of solutions produced by it, may depend crucially on the ability to effectively use this expertise. Ever since Sacerdoti's work on NOAH [19], the conventional wisdom in the planning community has held that non-primitive (abstract) actions, and action reduction schemas provide a flexible way of capturing the human expertise, as well as using it to control planning. Planners that use such action reduction (also called "task reduction") schemas have come to be known as "hierarchical task network (HTN) planners."

HTN planners, such as SIPE [22], O-Plan [7] and VICAR [5] have been used in several industrial applications. However, much of the formalization of planning algorithms has been confined to traditional action-based planning approaches such as plan-space and state-space planning. The few existing formalizations of HTN planning, such as [8; 2], tend to start from scratch instead of building on top of the well-understood action-based formalizations. These suffer from two inherent disadvantages: First, they make it harder to transfer the progress being made in scaling-up traditional planners (e.g [3; 15; 14]) to hierarchical planning. Second, perhaps more important, most real-world domains tend to be "partially hierarchical" in that human expertise, in the form of task reduction knowledge, exists for only some parts of the domain. In such domains, the planner is forced to employ a hybrid approach of using the reduction knowledge where available, and defaulting to primitive actions for the otherwise.

Providing a principled framework for hybrid HTN and action-based planning is the main objective of this paper. To show how HTN planning can co-exist with action-based planning, we extend the unified refinement planning framework that we had developed earlier for action-based planning [11; 12; 13] to cover HTN planning. Our existing framework covers the full spectrum of action-based planning approaches by characterizing plan-space and state-space planners as independent refinements operating on a uniform plan representation [12]. In order to include HTN planning in this framework, we need to consider non-primitive actions and their reduction schemas as part of the domain specification, and generalize the plan-space and state-space refinements to handle plans containing non-primitive actions. We shall illustrate the issues involved in refining such plans by generalizing the plan-space refinement to handle non-primitive actions. Specifically, the plan-space refinement will be modified to introduce non-primitive actions in addition to primitive actions during precondition establishment, and handle them during

*This research is supported in part by an NSF young investigator award (NYI) IRI-9457634, and ARPA/Rome Laboratory planning initiative grants F30602-93-C-0039 and F30602-95-C-0247. We thank Tony Barrett, Reiko Tsuneto, Austin Tate and David Wilkins for critical comments on a previous draft. Copyright(c) 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

conflict resolution. The non-primitive actions thus introduced can be replaced with the help of action reduction schemas at later stages.

We will see that the technical challenges in this hybridization include understanding and ensuring properties such as systematicity, completeness, preservation of user-intent, and parsimony of solutions. Developing HTN planning on top of the existing formalization of action-based planning allows us to put into perspective the many surface differences between HTN and action-based planners (including the differing notions of completeness, the need for phantom establishments, and the difficulties in conflict resolution). Such a development could also support transfer of progress between action-based planning and HTN planning.

The rest of this paper is organized as follows: In the next section, we review the refinement planning framework developed for action-based planning approaches. Section 3 introduces HTN planning, and shows how the action-based refinement planning framework is enriched to allow non-primitive actions and their reduction. Section 4 shows how plan-space refinement can be extended to handle non-primitive actions. In Section 5, we discuss the strengths of our hybrid approach, and relate it to previous work. Section 6 presents the conclusions.

2 Preliminaries of Action-based Refinement planning

A planning problem is a 3-tuple $\langle I, G, \mathcal{A} \rangle$, where I is the complete description of an initial state, G is the (partial) description of the goal state, and $\mathcal{A}$ is a set of actions (also called "operators"). An action sequence S is said to be a solution for the planning problem, if S can be executed from I, and the resulting state of the world implies G.

A partial plan $\mathcal{P}$ is a 5-tuple $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞ corresponding to the beginning and ending points of the plan. $\mathcal{ST}$ maps step names to actions. Actions are modeled in the familiar STRIPS/ADL representation [18] with precondition and effect lists. O is a set of ordering constraints among the steps in T, which includes: "precedence constraints" of the form "$t_i \prec t_j$" that require t_i to precede t_j with other steps possibly intervening between them, and "contiguity constraints" of the form "$t_i * t_j$" that require t_i to come *immediately* before t_j. $\mathcal{B}$ is a set of binding constraints on the variables appearing in the preconditions and post-conditions of the actions. $\mathcal{L}$ is a set of auxiliary constraints that involve statements about truth of specific conditions over particular time intervals. Two important types of auxiliary constraints are: (1) *Interval Preservation Constraints (IPCs)* of the form "$(t \overset{p}{-} t')$" which demand that a condition p be preserved between steps t and t', and (2) *Point Truth Constraints (PTCs)* of the form "$\overset{p}{\rightarrow} t$" which demand that a condition p be necessarily true [4] in the situation before the step t.

```
Algorithm UCP(𝒫) /*Returns refinements of 𝒫 */

0. Termination Check: If a minimal candidate of 𝒫 is a solution
     to the problem, return it and terminate.
1. Progressive Refinement: Select and apply a (progressive) re-
     finement. Examples include forward state space, backward
     state space or plan space refinement. (To include HTN plan-
     ning, the existing refinements need to be generalized to han-
     dle non-primitive actions; see Figure 4 and Section 4.1.)
2. Non-progressive (Tractability Refinements) (optional) Se-
     lect and apply one or more tractability refinements. These
     include pre-satisfaction (also called conflict resolution), pre-
     ordering and pre-positioning refinements. (To include HTN
     planning, we need to add a new tractability refinement called
     Pre-reduction, shown in Figure 3, and generalize the existing
     ones to handle non-primitive actions; see Secion 4.2.)
3. Consistency Check: (Optional) If the partial plan is inconsis-
     tent (i.e., has empty candidate set).
4. Recursive Invocation: Call UCP on the the refined partial plan
     (if it is not pruned).
```

Figure 1: UCP: A generalized algorithm template for classical planning

The semantics of partial plans are given in terms of "candidate sets." A candidate of a partial plan is an action sequence that is consistent with the action, ordering, binding and auxiliary constraints in the plan. For example, if a partial plan contains a single action a, then any action sequence that does not contain at least one instance of a can not belong to the candidate set of that plan (see [11; 13] for more details). The shortest length candidates of a plan are called its minimal candidates, and these correspond to the ground linearizations of the plan that satisfy all auxiliary constraints.

Figure 1 describes a generic refinement planning template. Each recursive invocation of the planner checks to see if one of the minimal candidates of the supplied plan is a solution to the problem. If it is, then the planning process terminates. If not, the current plan needs to be refined further. This involves selecting one of the state-space or plan-space refinements and applying them to the plan. Optionally, a set of tractability refinements are also applied [13]. Finally, the algorithm is invoked recursively on the resulting plans (after an optional consistency check to prune out inconsistent plans).

3 Extending the model to include HTN Planning

3.1 Introduction to HTN Planning

HTN planning problem can be seen as a generalization of the classical planning problem, where, in addition to the primitive actions in the domain, the domain writer also specifies a set of non-primitive actions, and provides a set of schemas for reducing the non-primitive actions into other primitive and non-primitive actions. As an example, consider the travel domain example shown in Figure 2. Here, in addition to the

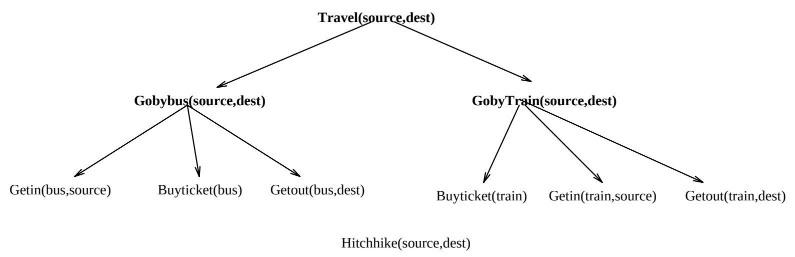

Figure 2: Schematic description of the actions and their relations in a travel domain. See the text for formal description of reduction schemas.

primitive actions, *Getin, Buyticket, Getout*, and *Hitch-hike*, we also have non-primitive actions *Gobybus*, *Gobytrain*, and *Travel*. Moreover, the action *Travel* can be reduced to either *Gobybus* or *Gobytrain*, and the action *Gobybus* in turn can be reduced (see below) to a plan fragment containing actions *Getin, Buyticket* and *Getout*.

We can view the reduction schemas as encoding a user's (domain-writer's) prescriptions about what constitute good plans. Informally, the only acceptable solutions are those that not only achieve the top-level goals upon execution, but can also be parsed in terms of the non-primitive actions that are provided to support those top-level goals [2]. We also say that such solutions "respect (preserve) user-intent." As an example, in the travel domain, although hitch-hiking or traveling without tickets are possible ways of achieving an $At(x)$ goal, they can not be parsed in terms of the *Gobybus* or *Gobytrain* non-primitive actions that are provided as ways to support the $At(x)$ goal, and are thus not acceptable. The preceding also implies that the completeness of HTN planners cannot be judged with respect to the set of all action sequences, but rather only with respect to those sequences that can be parsed by the schemas. In particular, in travel domain, an HTN planner's failure to find a plan based on hitch-hiking cannot be seen as an indication of its lack of completeness.

Indeed, there are two distinct notions of completeness that are relevant for HTN planners: The first, that we shall call "**schema completeness**" ensures that the non-primitive actions and task reduction schemas cover all desirable solutions for all problems of interest. The second, which we call "**planner completeness**," ensures that the planner is able to return every solution that can be parsed in terms of the non-primitive actions provided for the corresponding top-level goals. While the planning algorithm can guarantee planner completeness, it is not responsible in any way for schema completeness (just as it is not responsible for the correctness of the action specifications given to it). In practice, schema completeness needs to be analyzed through interactive validation techniques, and can be quite tedious. Surprisingly, there is very little acknowledgement of the importance of this validation in the literature on applications of HTN planning; Chien's work on applying HTN planning to a NASA image processing domain is a refreshing exception [6].

3.2 Modeling schemas and action reduction

The only extension required to the partial plan representation described in Section 2 to handle HTN planning is to allow non-primitive actions into the plan. Specifically, the steps T in the partial plan $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$, will now be mapped to two types of actions: *primitive actions* which correspond to the usual executable actions, and *non-primitive (abstract) actions*. The non-primitive actions have similar precondition/effect structure as the primitive actions. A subset of the effects of a non-primitive action may be distinguished by the domain writer as its "**primary effects**," with the implicit intention that the action may only be used to support its primary effects (see the discussion of non-minimality in Section 4.1). Candidates of the plan, as well as the solutions of the planning problem will still be primitive action sequences.

Reduction Schemas: The domain specification links each non-primitive action o to a set of reduction schemas. Each reduction schema $\mathcal{S}_i$ can be seen as a 2-tuple: $\langle \mathcal{P}_i, \mathcal{M}_i^{\mathcal{L}} \rangle$ where $\mathcal{P}_i$ is the partial plan fragment which can replace o, and $\mathcal{M}_i^{\mathcal{L}}$ maps (redirects) the auxiliary constraints involving o into new auxiliary constraints involving steps of $\mathcal{P}_i$.

Consider, for example, the reduction schema for reducing the non-primitive action $Gobybus(Src, Dest)$ (referred to in Figure 2)

$$\left\langle \begin{array}{l} \mathcal{P}_i : \left\langle \begin{array}{l} T, \mathcal{ST} : \{t_1 : Getin(bus, Src), t_2 : Buyticket(bus) \\ \qquad t_3 : Getout(bus, Dest)\} \\ O : \{(t_1 \prec t_2)(t_2 \prec t_3)\} \\ \mathcal{L} : \{(t_1 \overset{In(bus)}{-} t_3), (t_2 \overset{have(busticket)}{-} t_3)\} \end{array} \right\rangle \\ \mathcal{M}^{\mathcal{L}} : \{(? \overset{have(money)}{-} t_2), (t_3 \overset{At(Dest)}{-} ?)\} \end{array} \right\rangle$$

This schema specifies that $Gobybus(Src, Dest)$ action can be replaced by the plan fragment containing three actions of getting into the bus, buying the ticket, and getting out of the bus at the destination. The plan fragment also contains IPCs specifying that the ticket should be kept throughout the journey and that the agent should stay in the bus throughout the journey. Finally, the mapping part of the schema states that any IPC incident on the $Gobybus(Src, Dest)$ action which preserves the condition $have(money)$ should be redirected to the $Buyticket(bus)$ action of the new plan fragment (since the identity of the source of the IPC is not known until reduction time, it is denoted by "?").

Reduction of non-primitive actions: The procedure in Figure 3 describes the process of replacing a non-primitive action using a reduction schema. Consider a plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$ containing an abstract action t. Let $\mathcal{S} : \langle \mathcal{P}' : \langle T', O', \mathcal{B}', \mathcal{ST}', \mathcal{L}'\rangle, \mathcal{M}^{\mathcal{L}}\rangle$ be a reduction schema for t.

The partial plan that results from the reduction of the action t in $\mathcal{P}$ with the reduction schema $\mathcal{S}$ is denoted by $Reduce(\mathcal{P}, t, \mathcal{S})$, and is computed as follows:

$$Reduce(\mathcal{P}, t, \mathcal{S}) = \mathcal{P}_R : \left\langle \begin{array}{l} \{(T-t) \cup T'\}, \\ \{(O-O_t) \cup O' \cup O_m\}, \\ \{\mathcal{B} \cup \mathcal{B}' \cup \mathcal{B}'_m\}, \\ \{\mathcal{ST} \cup \mathcal{ST}'\}, \\ \{(\mathcal{L}-\mathcal{L}_t) \cup \mathcal{L}' \cup \mathcal{M}(\mathcal{L}_t)\} \end{array} \right\rangle$$

Where O_t and $\mathcal{L}_t$ are the ordering and auxiliary constraints involving t. These are replaced by the redirected constraints O_m and $\mathcal{M}(\mathcal{L}_t)$ during reduction. Note that in replacing t with its reduction, we need to redirect any constraints that explicitly name t, to steps in its reduction. The redirection of IPCs is done by the mapping $\mathcal{M}$ specified by the reduction schema. For the ordering constraints, the redirection is done automatically as follows. When t is reduced, the last step(s) of $\mathcal{P}'$ (i.e., those steps that do not precede any other steps in $\mathcal{P}'$) are forced to precede all the steps t' such that $t \prec t'$ before the reduction. Similarly, the first step(s) of $\mathcal{P}'$ are forced to follow all steps t'' such that $t'' \prec t$ before the reduction. Contiguity constraints are redirected in a similar manner.

Candidate set Semantics: We now explain how non-primitive actions constrain the candidate set of a plan. Let o be a non-primitive action, and let C_o be the set of all partial plans containing only primitive actions (called concretizations) that are obtained by starting with o and repeatedly reducing non-primitive actions using the user-supplied reduction schemas. If o is present in a partial plan $\mathcal{P}$, then the candidate set of $\mathcal{P}$ cannot contain any action sequence that does not satisfy the constraints corresponding to at least one concretization in C_o. Non-primitive actions can thus be seen as "*disjunctive constraints*" on the partial plan. The process of reducing non-primitive actions, described below, can be seen as making the implicit disjunction explicit, and pushing it into the search space. Consequently, reduction can be seen as a "tractability refinement" in the refinement planning terminology [13; 12].

Keeping track of partial hierarchicalization: Non-primitive actions and their associated reduction schemas are the only additions needed to the problem specification if the domain is fully hierarchicalized. If on the other hand, the domain is only partially hierarchicalized, then the domain writer needs a way of informing the planner as to which aspects of the domain are hierarchicalized. Although there are potentially many ways in which the hierarchicalization may be incomplete, we shall assume in this paper that the incompleteness is characterized with respect to specific conditions, using assertions of type "$hierarchicalized(condition)$." In particular, the fact that a condition c is fully hierarchicalized is denoted by $hierarchicalized(c)$. It means that all desirable plans for achieving the condition c can be parsed in terms of a non-primitive action whose primary effects include c. A fully hierarchicalized domain is denoted in shorthand by $hierarchicalized(-)$.[1] This approach of keeping track of hierarchicalization of a domain in terms of individual conditions is akin to keeping track of completeness of a knowledge base with localized closed world assumptions [10].

Algorithm Pre-reduce($\mathcal{P}$)

1. **Action Selection:** Pick an unreduced action $t \in T$ from $\mathcal{P}$ to work on. *Not a backtrack point.*
2. **Action Reduction:** Non-deterministically select a reduction schema $S{:}\,\mathcal{P}'$ for reducing t. Replace t in $\mathcal{P}$ with $\mathcal{P}'$ (This involves removing t from $\mathcal{P}$, merging the step, binding, ordering, symbol table and auxiliary constraints fields of $\mathcal{P}'$ with those of $\mathcal{P}$, and redirecting the ordering and auxiliary constraints in $\mathcal{P}$ which refer to t so that they refer to elements of $\mathcal{P}'$).
   Backtrack point; all reduction possibilities must be considered

Figure 3: Pre-reduction Refinement. Can be called as a tractability refinement at step 2 in Figure 2

Extending the refinement planning template: To extend the refinement planning template in Figure 1 to handle HTN planning, we need to add the pre-reduction refinement (see Figure 3) as one of the tractability refinements in step 2, and need to generalize the action-based progressive and tractability refinements in steps 1 and 2 to handle non-primitive actions.[2]

In the next section, we will illustrate the issues involved in this generalization by considering plan space and pre-satisfaction (conflict resolution) refinements. Our choice of plan-space refinement for illustration purposes is motivated by the fact that most implemented HTN planners can be seen as using such generalized plan-space refinements.

4 Generalizing refinements to consider non-primitive actions

4.1 Generalizing Plan Space Refinement

Traditional plan-space (PS) refinement involves adding constraints to support a pre-requisite $c@t$ (where c is required to be true at step t) with the help of the effects of a new or exist-

[1] It is possible to have a finer characterization of the hierarchicalization of a condition, by allowing the domain writer to specify a context in which the hierarchicalization of c is complete. The context may include information about sibling goals of c, as well as properties of the initial state.

[2] Another way of including HTN planning into the template in Figure 1 would be to leave all refinements as they are, and add an extra filter to the consistency check to prune partial plans that cannot be parsed in terms of the non-primitive actions. This approach, that we can call "bottom-up" HTN planning, has been advocated by Barrett and Weld [2].

ing step t'. As part of the establishment, t' is constrained to precede t, and is forced to give c (if c is a conditional effect, this involves adding the causation precondition of c as a precondition to t' [18]). Optionally, IPCs of the form $(t' \overset{c}{-} t)$ are posted to preserve the condition c between t' and t. The HPS refinement, which is a generalized version of the PS refinement to handle non-primitive actions, is shown in Figure 4.[3] We shall explain its development from the PS refinement in the following.

Consider the travel domain of Figure 2. Suppose our top-level goal is $At(SF)$, and we have $At(Phx)$ in the initial state. Six of the ten actions in the domain – *Travel(Phx,SF), Gobybus(Phx,SF), GobyTrain(Phx,SF),Getout(bus,SF), Getout(train,SF),* and *Hitchhike(car,SF)* – are all capable of achieving this condition. Which of these actions should HPS refinement consider in its different establishment branches?

We note that if HPS refinement considers only the primitive actions, it will correspond to the traditional plan-space refinement. Such a version may however violate the user-intent inherent in the schema specification (as it will for example, permit plans that involve ticket-less travel, which does not respect the intent of the schemas in Figure 2). If we generate branches corresponding to non-primitive actions along side branches for all primitive actions (as is done for example in DPOCL [24]), we will have a redundant search space. As an example, if HPS refinement considered all six actions as establishment possibilities in our example, then two of its refinements would be $\mathcal{P}_1 : t_0 \prec Gobybus(Phx, SF) \prec t_\infty$ and $\mathcal{P}_2 : t_0 \prec Getout(bus, SF) \prec t_\infty$. Since the non-primitive action $Gobybus(Phx, SF)$ ultimately gets reduced into a plan containing the primitive action $Getout(bus, SF)$, the plans have overlapping candidate sets, thus making HPS refinements non-systematic.

To avoid redundancy in the search space, and to respect user-intent, we use the convention that a goal should be established by considering the dominance relations between actions. We consider an action t to be **dominated** by another action t', if there exists a reduction of t' that contains t. Dominance relation is asymmetric except in those cases where the reduction schemas contain recursion. Such dominance relations can be computed easily from an action hierarchy graph such as the one shown in Figure 2. In the travel domain, the action $Gobybus(Phx, SF)$ dominates $Getout(Phx, SF)$ and is dominated by $Travel(Phx, SF)$.

Considering undominated actions to ensure systematicity: Assuming that we are using a systematic plan-space refinement as the basis for HPS, we can ensure that the plans generated by HPS will have non-overlapping candidate sets by requiring that none of the actions considered for the establishment of a given goal are dominated by the other actions being considered for the same goal. Thus we cannot consider $Travel(Phx, SF)$ and $Gobybus(Phx, SF)$ together as sibling choices in establishment *if* we want systematicity.

[3] Although the algorithm in the figure enforces a precedence relation between t' and t, if either one or both of them are non-primitive, then such an ordering could be too conservative. We really only need to add an ordering between the eventual action in the concretization of t' that gives c and the corresponding action in the concretization of t that requires c. Such an ordering will need to be re-directed incrementally as t' and t are being reduced. However, most implementations and formalizations of HTN planning retain the conservative orderings to avoid having to deal with redirection of ordering constraints.

Algorithm Refine-plan-hybrid-plan-space ($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */

1. **Goal Selection:** Pick an open prerequisite $c@t$ from $\mathcal{P}$ to work on. *Not a backtrack point.*
2. **Goal Establishment:** Non-deterministically select a new or existing establisher step t' for $c@t$ which is capable of giving c. If t' is a *new step*, make sure that it is *maximally abstract* with respect to c (i.e., *there is no non-primitive action t'' which can also give c and which dominates t'*). If t' is also non-primitive, make sure that c is a primary effect of t'. Introduce enough constraints into the plan such that (i) t' will precede t (ii) t' will have an effect c, and (iii) c will persist until t. *Backtrack point; all establishers need to be considered.*
3. **Bookkeeping:** (Optional) Add interval preservation constraints of the form $(t' \overset{c}{-} t)$ and $(t' \overset{\neg c}{-} t)$ noting the establishment decisions, to ensure that these decisions are not violated or duplicated by the latter refinements.
4. **Phantom Establishment:** If $\mathcal{P}$ contains non-primitive actions, then in addition to the plans generated above, consider also $\mathcal{P}'$ which is $\mathcal{P}+ \overset{c}{\rightarrow} t$. (The PTC $\overset{c}{\rightarrow} t$ in $\mathcal{P}'$ will never be considered for establishment explicitly. However, any solution derived from $\mathcal{P}'$ must satisfy it [12]).

Figure 4: Plan Space Refinement Modified to work in the presence of non-primitive actions. This can be called as a progressive refinement at step 1 of the procedure in Figure 2.

Considering maximally abstract actions to respect user-intent: An action t is said to be *maximally abstract* with respect to a condition, if it is not dominated by any other action giving that condition. We will respect user intent if we ensure that all the actions considered for establishment of a condition are maximally abstract with respect to that condition. Thus, we will select $Travel(Phx, SF)$ action rather than the $Gobybus(Phx, SF)$ or $Gobytrain(Phx, SF)$ actions to establish $At(SF)$ condition.

The maximal abstractness restriction applies only to step-addition and not to simple establishment, since once a primitive action capable of giving the condition is already in the plan, we may as well use it.

Handling primitive actions during establishment: In satisfying the "maximal abstractness" restriction, there is a question as to whether or not the HPS refinement should consider primitive actions which can establish some condition c, but are not dominated by any other actions (e.g. the $Hitchhike$ action in the travel domain example). If the domain specification contains $hierarchicalized(c)$ (see Section 3), then no primitive actions need to be considered for any goals that are also fulfilled by non-primitive actions. Otherwise, we will allow un-dominated primitive actions along side non-primitive actions. To prefer solutions that can be parsed by existing schemas where possible, we would want to bias the search such that establishment branches corresponding to non-primitive actions are preferred over the branches corresponding to primitive ones.

Use of primary effects to avoid non-minimal plans: Another potential concern is that selecting maximally abstract actions to establish a goal may lead us into scenarios where a complex plan is being executed only because the agent is interested in a secondary side effect of some non-primitive action in that plan. For example, suppose the planner has a single top-level goal "$have(trainticket)$" (presumably because the agent is an avid ticket collector). The action $Buytrainticket()$ will give the condition, and so presumably does the more abstract action $Gobytrain(x, y)$. Since the latter dominates the former, maximal abstract step restriction would require us to select the $Gobytrain()$ action to achieve $have(trainticket)$, which leads to a non-minimal plan for the agent.

We punt this problem by using only primary effects (see Section 3.2) during establishment with non-primitive actions, and assuming that the domain writer has specified the primary effects of non-primitive actions with appropriate care. Thus, if the domain writer listed on $At(x)$ as a primary effect of $Gobytrain()$ action, then we would not consider it to establish $have(Trainticket)$ goal. Our use of primary effects here is similar to (and explains the motivation behind) the "to-do" and "use-only-for" slots in the Nonlin [21] and O-Plan [7] reduction schemas, and the "purpose" slots in SIPE's [22] reduction schemas.

Phantom Establishments: Since non-primitive actions do not advertise all the possible effects of the actions involved in their eventual reductions, their presence necessitates a change to the "simple-establishment" phase (wherein the pre-requisite is established by the effects of steps currently existing in the plan) of the action-based plan-space refinement. Specifically, even if none of the explicit effects of the existing steps support a pre-requisite $c@t$, if one of the steps t' in the plan corresponds to a non-primitive action, then one of the primitive actions resulting from the eventual reduction of t' may very well have an effect c (and can thus support $c@t$). If we do not account for this possibility, we can lose completeness (since plan-space refinement considers each pre-requisite only once, i.e., does not backtrack on the goal order).

In order to handle this scenario, we provide an additional way of establishing the $c@t$ pre-requisite, viz., converting it into a point truth constraint "$\xrightarrow{c} t$." This process, called "phantomization," essentially leaves open a back door for supporting $c@t$ with the effects of actions that come into the plan through reduction (since the only way this PTC can be satisfied eventually is if c becomes necessarily true at t because of the effects of the actions present in the plan at that time).[4]

4.2 Conflict resolution with non-primitive actions

In normal action-based planning, the need for conflict resolution arises if the plan has an IPC $(t_1 \overset{p}{-} t_2)$ and a step t_3 which has a potential effect $\neg p$. The conflict is resolved by posting the constraints to ensure that either t_3 precedes t_1, or it follows t_2, or t_3 will not give $\neg p$. Generalizing conflict resolution with non-primitive actions presents some problems. Suppose t_1, t_2 and t_3 in the above example are all non-primitive. To avoid this conflict, we need to only ensure that the specific primitive step t_3^p, which actually deletes p in the eventual reduction of t_3 should come outside the interval between t_1^p and t_2^p, where the latter two are the primitive steps resulting from reduction of t_1 and t_2 that add and need p respectively. Since the identity of t_3^p, t_1^p and t_2^p will not be known until the three non-primitive actions are completely reduced, one cannot post simple ordering relations between these non-primitive actions to resolve the threat. Furthermore, even if the introduced ordering relations are found to be inconsistent with the existing constraints of the plan, it still doesn't mean that the conflict cannot be eventually resolved after further reduction [23].

In order to handle these problems, we require that conflict resolution be applied only with respect to IPCs and threats where the producer and consumer steps of the IPC as well as the threatening step are all primitive.

5 Discussion and related work

We have presented a hybrid planning approach that has the ability to consistently handle partially hierarchical domains. This ability supports incremental hierarchicalization of a domain: the domain-writer (or an autonomous learner) acting in concert with the planner may continue to provide additional reduction schemas making the domain fully hierarchicalized with respect to more conditions. Reduction schemas will be used for the parts of the problem for which they are present; and normal action-based planning will be used for the other parts in a seamless fashion. As the experience of the domain-writer/learner increases, more and more schemas may be written, tightening the control over the planner's search space.

The ability to handle partially hierarchicalized domains also removes one of the main criticisms of HTN planning from the reactive planning community (c.f. [20; 16])–viz., HTN planning reduces the flexibility of an agent to respond to situations that are not anticipated by the writer of the task-reduction schemas. It may thus increase the acceptance of HTN approaches in that community.

[4] It is possible to reduce the number of phantom establishment branches, if we keep track of the "possible" effects of non-primitive actions. A condition c is a possible effect of a non-primitive action a, if c is an effect of a or any action that is dominated by a. We can avoid generating phantom establishment branches for condition c if none of the non-primitive actions in the plan have c as a possible effect.

Developing HTN planning on top of the existing formalization of action-based planning confers us two advantages. First, it allows us us to put into perspective the many surface differences between HTN and action-based planners, such as the need for phantomization, and the differing notions of completeness. Second and perhaps more important, our work also helps us exploit recent progress on action-based planners, including SATPLAN [15] and Graphplan [3], in the context of HTN planners [14]. Indeed, some of our ongoing work involves adapting the planning as satisfiability approach to HTN planning [17]. As in this paper, our HTN encodings are obtained by generalizing the encodings for action-based planning.

In theory, it is also possible to represent partially hierarchical domains in existing HTN planners such as SIPE [22] and O-Plan [7], as well as in the existing HTN formalizations [8; 2]. For every undominated primitive action, we simply introduce a non-primitive action and a degenerate reduction schema that reduces it to the primitive action. What is novel about our framework is that we explicitly consider the issues involved in ensuring systematicity, and preserving user-intent in such partially hierarchical domains.

Some previous systems, including IPEM [1] and DPOCL [24], attempted to explicitly combine task reduction and plan-space planning. Neither of these approaches however guarantees an integration that ensures systematicity and preservation of user-intent. For example, DPOCL allows for the establishment of a precondition both with a non-primitive action and a primitive action which it dominates. As we argued earlier, this can lead to a loss of systematicity as well as violation of user-intent. Our approach also provides clear semantics of plans containing non-primitive actions, and thus explains the essential differences between the completeness results of plan-space and HPS refinements.

Another research effort that allows combination of HTN and plan-space planning approaches is that of Chien and his co-workers [5]. They argue that the HTN and plan-space techniques should be kept separate, with the former working on "activity" goals while the latter work on "state" goals. Domain information pertaining to these techniques is also kept separate. While our approach allows for such a separation, it also allows for HTN and plan-space approaches to work together on the same class of goals in a principled way.

6 Conclusion

In this paper, we have argued for hybrid planning approaches that combine HTN and action-based planning algorithms. Such approaches facilitate planning in partially hierarchical domains. We presented such a hybrid planning framework by extending our existing refinement planning framework for action-based planning [11; 13; 12]. The extensions involve considering non-primitive actions and their reduction schemas as part of the domain specification, and generalizing the traditional refinements to handle plans containing non-primitive actions. We illustrated the issues involved in such generalization in the context of plan-space and conflict resolution refinements. We discussed what it takes for such a hybrid approach to respect user-intent, while guaranteeing systematicity and completeness. We have also discussed the relation between our hybrid framework and previous attempts to formalize HTN planning. Developing HTN planning on top of the existing formalization of action-based planning allows us to put into perspective the many surface differences between HTN and action-based planners, and could help in transferring progress between HTN and action-based planning approaches. As an example of the latter, we recently extended the SATPLAN [15] approach to HTN planning [17].

References

[1] J.A. Ambros-Ingerson and S. Steel. Integrating Planning, Execution and Monitoring. In *Proc. 7th AAAI*, 1988.

[2] A. Barrett and D. Weld. Task Decomposition via Plan Parsing. In *Proc. AAAI-94*.

[3] A. Blum and M. Furst, Fast planning through planning graph analysis, Proc. of IJCAI-95, 1636-1642.

[4] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[5] S. Chien, T. Estlin and X. Wang. An argument for hybrid HTN/Operator Planning. In Proc. 4th European Conference on Planning. 1997.

[6] S. Chien. Static and Completion analysis for planning knowledge base development and verification. In *Proc. AIPS-96*, 1996.

[7] K. Currie and A. Tate. O-Plan: The Open Planning Architecture. *Artificial Intelligence*, 51(1), 1991.

[8] K. Erol. Hierarchical task network planning: Formalization, Analysis and Implementation, Ph.D thesis, Dept. of computer science, Univ. of Maryland, College Park, 1995.

[9] K. Erol, J. Hendler, D.S. Nau and R. Tsuneto. A critical look at critics in HTN planning. In *Proc. IJCAI-95*, 1995.

[10] O. Etzioni, K. Golden and D. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1-2), 113–148.

[11] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A Unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* special issue on Planning and Scheduling. Vol. 76. 1995. and

[12] S. Kambhampati and B. Srivastava. Universal Classical Planner: An algorithm for unifying state space and plan space approaches. In New Trends in AI Planning: EWSP 95, IOS Press, 1995. (Extended version available as technical report ASU CSE 96-006 at http://rakaposhi.eas.asu.edu/ucp-tr.ps).

[13] S. Kambhampati, Refinement search as a unifying framework for plan synthesis, AI magazine, Summer 1997.

[14] S. Kambhampati. Challenges in bridging plan synthesis paradigms, Proc. IJCAI-97, 1997.

[15] H. Kautz and B. Selman, Pushing the envelope: Planning, Propositional logic and Stochastic search, Proc. of AAAI-96.

[16] P. Maes, Situated agents can have goals, Robotics and autonomous systems 6, 1990, 49-70.

[17] A. Mali and S. Kambhampati. Encoding HTN planning in propositional logic. Proc. 4th AI Planning Systems Conf., 1998.

[18] E.P.D. Pednault. Synthesizing Plans that contain actions with Context-Dependent Effects. *Computational Intelligence*, Vol. 4, 356-372 (1988).

[19] E. Sacerdoti. The nonlinear nature of plans. In *Proc. IJCAI-75*, 1975.

[20] R. Simmons, Structured control for autonomous robots, IEEE transactions on robotics and automation, Feb. 1994.

[21] A. Tate. Generating Project Networks. In *Proceedings of IJCAI-77*, pages 888–893, Boston, MA, 1977.

[22] D. Wilkins. *Practical Planning*. Morgan Kaufmann (1988).

[23] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence Journal*, 6:12–24, 1990.

[24] R.M. Young, M.E. Pollack and J.D. Moore. Decomposition and Causality in Partial-Order Planning. In *Proc. 2nd Intl. Conf. on AI Planning Systems*, 1994.