UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning*

Kutluhan Erol kutluhan@cs.umd.edu
James Hendler hendler@cs.umd.edu
Dana S. Nau nau@cs.umd.edu

Computer Science Department,
Institute for Systems Research and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

One big obstacle to understanding the nature of hierarchical task network (HTN) planning has been the lack of a clear theoretical framework. In particular, no one has yet presented a clear and concise HTN algorithm that is sound and complete. In this paper, we present a formal syntax and semantics for HTN planning. Based on this syntax and semantics, we are able to define an algorithm for HTN planning and prove it sound and complete.

Introduction

In AI planning research, planning practice (as embodied in implemented planning systems) tends to run far ahead of the theories that explain the behavior of those planning systems. For example, STRIPS-style planning systems[1] were developed more than twenty years ago (Fikes *et al.* 1971), and most of the practical work on AI planning systems for the last fifteen years has been based on hierarchical task-network (HTN) decomposition (Sacerdoti 1977; Tate 1990; Wilkins 1988a; Wilkins 1988b). In contrast, although the past few years have seen much analysis of planning using STRIPS-style operators, (Chapman 1987; McAllester *et al.* 1991; Erol *et al.* 1992b; Erol *et al.* 1992a), there has been very little analytical work on HTN planners.

One big obstacle to such work has been the lack of a clear theoretical framework for HTN planning. Two recent papers (Yang 1990; Kambhampati *et al.* 1992) have provided important first steps towards formalizing HTN planning, but these focused on the syntax, rather than the semantics. As a result, no one has presented a clear and concise HTN algorithm that is sound and complete. In this paper, we present exactly such an algorithm.[2]

HTN planning

In HTN planning, the world and the basic actions that can be performed are represented in a manner similar to the representations used in STRIPS (Fikes *et al.* 1971; Chapman 1987). Each "state" of the world is represented as a collection of atoms, and operators are used to associate effects to actions (primitive tasks). The fundamental difference between STRIPS-style planning and HTN planning is the representation of "desired change" in the world.

HTN-planning replaces STRIPS-style goals with tasks and task networks, which are provably more powerful (Erol *et al.* 1994c; Erol *et al.* 1994b). There are three types of tasks. *Goal tasks*, like goals in STRIPS, are properties we wish to make true in the world, such as having a house. *Primitive tasks* are tasks we can directly achieve by executing the corresponding action, such as moving a block, or turning a switch on. *Compound tasks* denote desired changes that involve several goal tasks and primitive tasks; e.g., building a house requires many other tasks to be performed (laying the foundation, building the walls, etc.). A compound task allows us to represent a "desired change" that cannot be represented as a single goal task or primitive task. For example, the compound task of "building a house" is different from the goal task of "having a house," since buying a house would achieve the goal task, but not the compound task. As another example, consider the compound task of making a round-trip to New York. This could not be easily expressed as a single goal task either, since the initial and final states would be the same.

*This work was supported in part by NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center and NSF grant IRI-8907890 and ONR grant N00014-91-J-1451 to the University of Maryland Computer Science Department.

[1] We will refer to planning systems that use STRIPS operators (with no decompositions) as STRIPS-style planners, ignoring algorithmic differences among them that are not relevant to the current work.

[2] Due to space limitations, we cannot include the details of our proofs, nor the details of how our work compares to (Sacerdoti 1977; Tate 1990; Wilkins 1988a; Yang 1990; Kambhampati *et al.* 1992). These are presented in (Erol *et al.* 1994a).

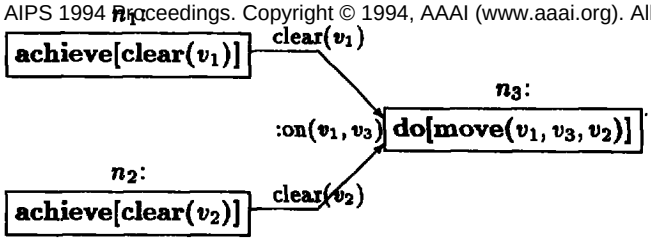

Figure 1: Graphical representation of a task network.

Tasks are connected together in HTN planning via the use of *task networks*, which are also called "procedural nets" in some of the literature (Sacerdoti 1977; Drummond 1985). Figure 1 gives a graphical representation of a task network containing three tasks: clearing the block v_1, clearing the block v_2, and moving v_1 to v_2. It also shows the conditions that moving v_1 should be done last, v_1 and v_2 should remain clear until we move v_1, and that the variable v_3 is bound to the location of v_1 before v_1 is moved.

A number of different systems that use heuristic algorithms have been devised for HTN planning (Tate 1990; Vere 1983; Wilkins 1988a), and several recent papers have tried to provide formal descriptions of these algorithms (Yang 1990; Kambhampati *et al.* 1992). Figure 2 presents the essence of these algorithms. As shown in this figure, HTN planning works by expanding tasks and resolving conflicts iteratively, until a conflict-free plan can be found that consists only of primitive tasks.

Expanding each non-primitive task (steps 3–5) is done by choosing an appropriate reduction, which specifies one possible way of accomplishing that task. Reductions are stored as *methods*, which associate non-primitive tasks with task networks. For example, in the blocks world, the non-primitive task **achieve[on(v_1, v_2)]** might be associated with the task network shown in Figure 1.

The task network produced in step 5 may contain conflicts caused by the interactions among tasks. The job of finding and resolving these interactions is performed by critics. Historically speaking, critics were introduced into NOAH (Sacerdoti 1977) to identify, and deal with, several kinds of interactions (not just deleted preconditions) between the different networks used to reduce each non-primitive operator. This is reflected in steps 6 and 7 of Figure 2: after each reduction, a set of critics is checked so as to recognize and resolve interactions between this and any other reductions. Thus, critics provide a general mechanism for detecting interactions early, so as to reduce the amount of backtracking. For a more detailed discussion of the many different ways critic functions have been used, see (Tate *et al.* 1990).

Although the basic idea of HTN planning has been around since 1974, the lack of a clear theoretical foundation has made it difficult to explore its properties.

1. Input a planning problem **P**.
2. If **P** contains only primitive tasks, then resolve the conflicts in **P** and return the result. If the conflicts cannot be resolved, return failure.
3. Choose a non-primitive task t in **P**.
4. Choose an expansion for t.
5. Replace t with the expansion.
6. Use critics to find the interactions among the tasks in **P**, and suggest ways to handle them.
7. Apply one of the ways suggested in step 6.
8. Go to step 2.

Figure 2: The basic HTN Planning Procedure.

In particular, although it is easy to state this algorithm, proving it sound and complete requires considerable formal development. In the following section, we present a syntax and semantics for HTN planning.

HTN Formalism

Syntax

Our language $\mathcal{L}$ for HTN planning is a first-order language with some extensions, which generalizes the syntaxes of (Yang 1990; Kambhampati *et al.* 1992). The vocabulary of $\mathcal{L}$ is a tuple $\langle V, C, P, F, T, N \rangle$, where V is an infinite set of variable symbols, C is a finite set of constant symbols, P is a finite set of predicate symbols, F is a finite set of *primitive*-task symbols (denoting actions), T is a finite set of *compound*-task symbols, and N is an infinite set of symbols used for labeling tasks. P, F, and T are mutually disjoint.

A *primitive task* is a syntactic construct of the form $do[f(x_1, \ldots, x_k)]$, where $f \in F$ and $x_1, \ldots, x_k$ are terms. A *goal task* is a syntactic construct of the form $achieve[l]$, where l is a literal. A *compound task* is a syntactic construct of the form $perform[t(x_1, \ldots, x_k)]$, where $t \in T$ and $x_1, \ldots, x_k$ are terms. We sometimes refer to goal tasks and compound tasks as non-primitive tasks.

A *task network* is a syntactic construct of the form $[(n_1 : \alpha_1) \ldots (n_m : \alpha_m), \phi]$, where

- each α_i is a task;
- $n_i \in N$ is a label for α_i (to distinguish it from any other occurrences of α_i in the network);
- ϕ is a boolean formula constructed from variable binding constraints such as $(v = v')$ and $(v = c)$, temporal ordering constraints such as $(n \prec n')$, and truth constraints such as (n, l), (l, n), and (n, l, n'), where $v, v' \in V$, l is a literal, $c \in C$, and $n, n' \in N$.[3]

[3] We also allow n, n' to be of the form $first[n_i, n_j, \ldots]$ or $last[n_i, n_j, \ldots]$ so that we can refer to the task that starts first and to the task that ends last among a set of tasks, respectively.

$((n_1 : achieve[clear(v_1)])(n_2 : achieve[clear(v_2)])$
$(n_3 : do[move(v_1, v_3, v_2)])$
$(n_1 \prec n_3) \wedge (n_2 \prec n_3) \wedge (n_1, clear(v_1), n_3)$
$\wedge (n_2, clear(v_2), n_3) \wedge (on(v_1, v_3), n_3)$
$\wedge \neg(v_1 = v_2) \wedge \neg(v_1 = v_3) \wedge \neg(v_2 = v_3))$

Figure 3: Formal representation of the task network of Fig. 1.

Intuitively (this will be formalized in the "Semantics" section), $(n \prec n')$ means that the task labeled with n must precede the one labeled with n'; (n, l), (l, n) and (n, l, n') mean that l must be true immediately after n, immediately before n, and between n and n', respectively. Both negation and disjunction are allowed in the constraint formula.

Figure 3 gives a formal representation of the task network shown in Figure 1. A *primitive task network* is a task network that contains only primitive tasks.

A *planning operator* is a syntactic construct of the form $(f(v_1, \ldots, v_k), l_1, \ldots, l_m)$, where $f \in F$ is a primitive task symbol, $l_1, \ldots, l_m$ are literals denoting the primitive task's effects (also called postconditions), and $v_1, \ldots, v_k \in V$ are the variables appearing in $l_1, \ldots, l_m$. Our operators do not contain explicit STRIPS-style preconditions; these are realized in task networks. The conditions the planner must actively make true are realized as goal tasks, and the conditions the planner must only verify to hold[4] are realized as truth constraints. For example, consider the task network for achieving $on(v_1, v_2)$ in blocks world, shown in Fig. 1. To move the block v_1, first we need to make it clear, and this precondition is expressed as a goal task. Now, consider the condition $on(v_1, v_3)$. Its purpose is to ensure that v_3 is the block under v_1; we certainly do not intend to move v_1 onto v_3. Thus, it is expressed by the constraint $(on(v_1, v_3), n_3)$ on the task network.

A *method* is a construct of the form (α, d) where α is a non-primitive task, and d is a task network. As we will define formally in the "Semantics" seciton, this construct means that one way of accomplishing the task α is to accomplish the task network d, i.e. to accomplish all the subtasks in the task network without violating the constraint formula of the task network. To accomplish a goal task ($achieve[l]$), l needs to be true in the end, and this is an implicit constraint in all methods for goal tasks. If a goal is already true, then an empty plan can be used to achieve it. Thus, for each goal task, we (implicitly) have a method $(achieve[l], [(n : do[f]), (l, n)])$ which contains only one dummy primitive task f with no effects, and the constraint that the goal l is true immediately before $do[f]$.

A *planning domain* $\mathcal{D}$ is a pair $\langle Op, Me \rangle$, where Op is a list of operators (one for each primitive task), and Me is a list of methods. A *planning problem* $\mathbf{P}$ is a triple $\langle d, I, \mathcal{D} \rangle$, where $\mathcal{D}$ is a planning domain, I is the initial state, and d is the task network we need to plan for.

A *plan* is a sequence σ of ground primitive tasks. $solves(\sigma, d, I)$ is a syntactic construct which we will use to mean that σ is a plan for the task network d at state I.

Model-Theoretic Semantics

Before we can develop a sound and complete planning algorithm for HTN planning, we need a semantics that provides meaning to the syntactic constructs of the HTN language, which in turn would define the set of plans for a planning problem.

A semantic structure for HTN planning is a triple $M = \langle \mathcal{S}_M, \mathcal{F}_M, \mathcal{T}_M \rangle$; we omit the subscript M whenever it is clear from context which model we are referring to. The members of the triple are described below.

The set of states is $\mathcal{S} = 2^{\{\text{all ground atoms}\}}$. Each member of $\mathcal{S}$ is a state, consisting of the atoms true in that state. Any atom not appearing in a state is considered to be false in that state. Thus, a state corresponds to a "snapshot" instance of the world.

$\mathcal{F} : F \times C^* \times \mathcal{S} \rightarrow \mathcal{S}$ interprets the actions. Given an action symbol from F, with ground parameters from C, and an input state, $\mathcal{F}$ tells us which state we would end up with, if we were to execute the action.

$\mathcal{T} : \{\text{non-primitive tasks}\} \rightarrow 2^{\{\text{primitive task networks}\}}$ interprets each non-primitive task d as a (not necessarily finite) set of *primitive* task networks $\mathcal{T}(d)$. Each primitive task network d' in $\mathcal{T}(d)$ gives a set of actions that would achieve d under certain conditions (as specified in the constraint formula of d'). There are two restrictions on the interpretation of goal tasks of the form $achieve[l]$: (1) at the end of the task, l must be true, and (2) if l is already true, we must be able to achieve l by doing nothing.

Task networks can be interpreted similarly, and we extend the domain of $\mathcal{T}$ to cover task networks as follows:

- $\mathcal{T}(\alpha) = \{[(n : \alpha), TRUE]\}$, if α is a ground primitive task. Thus, in order to accomplish α, it suffices to execute it.
- $\mathcal{T}(d) = \{d\}$, if d is a ground primitive task network.
- To accomplish a non-primitive task network d, we need to accomplish each task in d without violating the constraint formula. Thus we define $\mathcal{T}(d)$ as follows. Let $d = [(n_1 : \alpha_1) \ldots (n_m : \alpha_m), \phi]$ be a ground task network possibly containing non-primitive tasks. Then

$$\mathcal{T}(d) = \{compose(d_1, \ldots, d_m, \phi) \mid \forall i \; d_i \in \mathcal{T}(\alpha_i)\},$$

where *compose* is defined as follows. Suppose

$$d_i = [(n_{i1} : \alpha_{i1}) \ldots (n_{ik_i} : \alpha_{ik_i}), \phi_i]$$

[4] These are generally called filter conditions in the literature

for each i. Then[5]

$$compose(d_1, \ldots, d_m, \phi) = [(n_{11} : \alpha_{11}) \ldots (n_{mk_m} : \alpha_{mk_m}), \phi_1 \wedge \ldots \phi_m \wedge \phi'],$$

where ϕ' is obtained from ϕ by making the following replacements:

- replace $(n_i < n_j)$ with $(last[n_{i1}, \ldots, n_{ik_i}] < first[n_{j1}, \ldots, n_{jk_j}])$;
- replace (l, n_i) with $(l, first[n_{i1}, \ldots, n_{ik_i}])$;
- replace (n_i, l) with $(last[n_{i1}, \ldots, n_{ik_i}], l)$;
- replace (n_i, l, n_j) with $(last[n_{i1}, \ldots, n_{ik_i}], l, first[n_{j1}, \ldots, n_{jk_j}])$;
- everywhere that n_i appears in ϕ in a $first[]$ or a $last[]$ expression, replace it with $n_{i1}, \ldots, n_{ik_i}$.

- $\mathcal{T}(d) = \bigcup_{d' \text{ is a ground instance of } d} \mathcal{T}(d')$, if d is a task network containing variables.

Satisfaction An operator $(t(x_1, \ldots, x_k), l_1, \ldots, l_m)$ is *satisfied* by a model M, if for any ground substitution θ on $x_1, \ldots, x_k$,and any state s,

$$\mathcal{F}_M(t, x_1\theta, \ldots, x_k\theta, s) = (s - \{l\theta \mid l \in \{l_1, \ldots, l_m\} \text{ is a negative literal}\}) \cup \{l\theta \mid l \in \{l_1, \ldots, l_m\} \text{ is a positive literal}\}.$$

Thus, a model M satisfies an operator, if M interprets the primitive task so that it has the corresponding effects.

Next, we want to define the conditions under which a model M satisfies $solves((\sigma, d, s)$, i.e., the conditions under which σ is a plan that solves the task network d starting at state s, with respect to the model M. First we consider the case where d is primitive.

Let $\sigma = (f_1(c_{11}, \ldots, c_{1k_1}), \ldots, f_m(c_{m1}, \ldots, c_{mk_m}))$ be a plan, d $= [(n_1 : \alpha_1) \cdots (n_{m'} : \alpha_{m'}), \phi]$ be a ground primitive task network, M be a model, and s_0 be a state. Let $s_i = \mathcal{F}_M(f_i, c_{i1}, \ldots, c_{ik_i}, s_{i-1})$ for $i = 1 \ldots m$ be the intermediate states. We define *a matching* π from d to σ to be a one to one relation from $\{1, \ldots, m'\}$ to $\{1, \ldots, m\}$ such that whenever $\pi(i) = j$, $\alpha_i = do[f_j(c_{j1}, \ldots, c_{jk_j})]$. Thus a matching provides a total ordering on the tasks. M satisfies $solves(\sigma, d, s)$ if $m = m'$, and there exists a matching π that makes the constraint formula ϕ true. The constraint formula is evaluated as follows:

- $(c_i = c_j)$ is true, if c_i, c_j are the same constant symbols;
- $first[n_i, n_j, \ldots]$ evaluates to $min\{\pi(i), \pi(j), \ldots\}$;
- $last[n_i, n_j, \ldots]$ evaluates to $max\{\pi(i), \pi(j), \ldots\}$;
- $(n_i \prec n_j)$ is true if $\pi(i) < \pi(j)$;
- (l, n_i) is true if l holds in $s_{\pi(i)-1}$;
- (n_i, l) is true if l holds in $s_{\pi(i)}$;
- (n_i, l, n_j) is true if l holds for all s_e, $\pi(i) \leq e < \pi(j)$;
- logical connectives $\neg, \wedge, \vee$ are evaluated as in propositional logic.

Let d be a task network, possibly containing non-primitive tasks. A model M satisfies $solves(\sigma, d, s)$, if for some $d' \in \mathcal{T}_M(d)$, M satisfies $solves(\sigma, d', s)$.

For a method to be satisfied by a given model, not only must any plan for d also be a plan for α, but in addition, any plan for a task network tn containing d must be a plan for the task network obtained from tn by replacing d with α. More formally, a method (α, d) is *satisfied* by a model M if the following property holds: given any plan σ, any state s, and any $d' \in \mathcal{T}_M(d)$, whenever there is a matching π such that σ at s satisfies the constraint formula of d', then there exists $d'' \in \mathcal{T}_M(\alpha)$ such that for some matching π' with the same range as π, σ at s makes the constraint formula of d'' true.

A model M satisfies a planning domain $\mathcal{D} = (Op, Me)$, if M satisfies all operators in Op, and all methods in Me.

Proof Theory

A plan σ solves a planning problem $\mathbf{P} = (d, I, \mathcal{D})$ if any model that satisfies $\mathcal{D}$ also satisfies $solves(\sigma, d, I)$. However, given a planning problem, how do we find plans that solve it?

Let d be a primitive task network (one containing only primitive tasks), and let I be the initial state. A plan σ is a *completion* of d at I, denoted by $\sigma \in comp(d, I, \mathcal{D})$, if σ is a total ordering of the primitive tasks in a ground instance of d that satisfies the constraint formula of d. For non-primitive task networks d, $comp(d, I, \mathcal{D})$ is defined to be $\emptyset$.

Let d be a non-primitive task network that contains a (non-primitive) node $(n : \alpha)$. Let $m = (\alpha', d')$ be a method, and θ be the most general unifier of α and α'. We define $reduce(d, n, m)$ to be the task network obtained from $d\theta$ by replacing $(n : \alpha)\theta$ with the task nodes of $d'\theta$, modifying the constraint formula ϕ of $d'\theta$ into ϕ' (as we did for *compose*), and incorporating $d'\theta$'s constraint formula. We denote the set of reductions of d by $red(d, I, \mathcal{D})$. Reductions formalize the notion of *task decomposition*.

Here are the two rules we use to find plans:

R1. If $\sigma \in comp(d, I, \mathcal{D})$, conclude $\sigma \in sol(d, I, \mathcal{D})$.

R2. If $d' \in red(d, I, \mathcal{D})$ and $\sigma \in sol(d', I, \mathcal{D})$, conclude $\sigma \in sol(d, I, \mathcal{D})$.

The first rule states that the set of plans that achieve a primitive task network consists of the completions of the task network; the second rule states that if d' is a reduction of d, then any plan that achieves d' also achieves d.

[5] The formula actually is slightly more complicated than what is shown, because the variables and node labels in each d_i must be renamed so that no common variable or node label occurs.

procedure UMCP:

1. Input a planning problem $\mathbf{P} = (d, I, \mathcal{D})$.
2. if d is primitive, then
   If $comp(d, I, \mathcal{D}) \neq \emptyset$, return a member of it.
   Otherwise return FAILURE.
3. Pick a non-primitive task node $(n : \alpha)$ in d.
4. Nondeterministically choose a method m for α.
5. Set $d := reduce(d, n, m)$.
6. Set $\Gamma := \tau(d, I, \mathcal{D})$.
7. Nondeterministically set $d :=$ some element of Γ.
8. Go to step 2.

Figure 4: **UMCP**: Universal Method-Composition Planner

Next, we define $sol(d, I, \mathcal{D})$, the set of plans that can be derived using **R1** and **R2**:

$$\begin{aligned} sol_1(d, I, \mathcal{D}) &= comp(d, I, \mathcal{D}) \\ sol_{n+1}(d, I, \mathcal{D}) &= sol_n(d, I, \mathcal{D}) \cup \\ &\quad \bigcup_{d' \in red(d, I, \mathcal{D})} sol_n(d', I, \mathcal{D}) \\ sol(d, I, \mathcal{D}) &= \cup_{n < \omega} sol_n(d, I, \mathcal{D}) \end{aligned}$$

Intuitively, $sol_n(d, I, \mathcal{D})$ is the set of plans that can be derived in n steps, and $sol(d, I, \mathcal{D})$ is the set of plans that can be derived in any finite number of steps. The following theorem states that $sol(d, I, \mathcal{D})$ is indeed the set of plans that solves $(d, I, \mathcal{D})$

Theorem 1 (Equivalence Theorem) *Given a task network d, an initial state I, and a plan σ, σ is in $sol(d, I, \mathcal{D})$ iff any model that satisfies $\mathcal{D}$ also satisfies $solves(\sigma, d, I)$.*

This theorem follows from the fact that $sol(d, I, \mathcal{D})$ is constructed such that it always contains only the plans for a task network d with respect to the minimum model. We prove the theorem by constructing a model M such that for any non-primitive task α, $\mathcal{T}_M(\alpha)$ contains the primitive task networks that can be obtained by a finite number of reduction steps from α. Then we prove M to be the minimum model satisfying $\mathcal{D}$.

Corollary 1 R1 *and* **R2** *are sound and complete.*

Since the set of plans that can be derived using **R1** and **R2** is exactly $sol(d, I, \mathcal{D})$, the corollary immediately follows from the equivalence theorem.

A Hierarchical Planning Procedure.

Using the syntax and semantics developed in the previous section, we can now formalize the HTN planning procedure that we presented in Figure 2. Figure 4 presents our formalization, which we call **UMCP** (for Universal Method-Composition Planner).

It should be clear that **UMCP** mimics the definition of $sol(d, I, \mathcal{D})$, except for Steps 6 and 7 (which correspond to the critics). As discussed before, HTN planners typically use their critics for detecting and resolving interactions among tasks (expressed as constraints) in task networks at higher levels, before all subtasks have been reduced to primitive tasks. By eliminating some task orderings and variable bindings that lead to dead ends, critics help prune the search space. In our formalism, this job is performed by the critic function τ. τ inputs an initial state I, a task network d, a planning domain $\mathcal{D}$ and outputs a set of task networks Γ. Each member of Γ is a candidate for resolving some[6] of the conflicts in d. We need to put two restrictions on τ to ensure that it functions properly and that **UMCP** is sound and complete:

1. If $d' \in \tau(d, I, \mathcal{D})$ then $sol(d', I, \mathcal{D}) \subseteq sol(d, I, \mathcal{D})$. Thus, any plan for d' must be a plan for d ensuring soundness.
2. If $\sigma \in sol_k(d, I, \mathcal{D})$ for some k, then there exists $d' \in \tau(d, I, \mathcal{D})$ such that $\sigma \in sol_k(d', I, \mathcal{D})$.
   Thus, whenever there is a plan for d, there is a plan for some member d' of $\tau(d, I, \mathcal{D})$. In addition, if the solution for d is no further than k expansions, so is the solution for d'. The latter condition ensures that τ does not create infinite loops by undoing previous expansions.

In contrast to the abundance of well understood STRIPS-style planning algorithms (such as (Fikes *et al.* 1971; Chapman 1987; Barett *et al.* 1992; Kambhampati 1992)), HTN planning algorithms have typically not been proven to be sound or complete. However, using the formalism in this paper, we can establish the soundness and completeness of the HTN planning algorithm **UMCP**.

Corollary 2 (Soundness) *Whenever* **UMCP** *returns a plan, it achieves the input task network at the initial state with respect to all the models that satisfy the methods and the operators.*

Corollary 3 (Completeness)
Whenever **UMCP** *fails to find a plan, there is no plan that achieves the input task network at the initial state with respect to all the models that satisfy the methods and the operators.*

These results follow directly from the equivalence theorem using the fact that **UMCP** directly mimics $sol()$. The restrictions on the critic function ensure that τ does not introduce invalid solutions and that it does not eliminate valid solutions.

Conclusions

One big obstacle to understanding the nature of hierarchical task network (HTN) planning has been the lack of a clear theoretical framework. In this paper, we have presented a formal syntax and semantics for HTN planning. Based on this syntax and semantics, we

[6] It might be impossible or too costly to resolve some conflicts at a given level, and thus handling those conflicts can be postponed.

have defined an algorithm for HTN planning, and have proved it to be sound and complete.

This formalism also enables us to do complexity analyses of HTN planning, to assess the expressive power of the use of HTNs in planning, and to compare HTNs to planning with STRIPS-style operators. For example, we have been able to prove that HTN planning, as defined here, is formally more expressive than planning without decompositions (Erol *et al.* 1994c; Erol *et al.* 1994b). We are working on a deeper complexity analysis of HTNs and towards an understanding of where the complexity lies.

Our semantics characterizes various features of HTN planning systems, such as tasks, task networks, filter conditions, task decomposition, and critics. We believe that this more formal understanding of these aspects of planning will make it easier to encode planning domains as HTNs and to analyze HTN planners. Furthermore, the definition for whether a given model satisfies a planning domain can provide a criterion for telling whether a given set of methods and operators correctly describe a particular planning domain. We are currently exploring these further.

Finally, we are starting to explore the order in which tasks should be expanded to get the best performance, and more generally, in which order all commitments (variable bindings, temporal orderings, choice of methods) should be made. This will involve both algorithmic and empirical studies. Our long term goals are to characterize planning domains for which HTN planning systems are suitable, and to develop efficient planners for those domains. Our framework provides the necessary foundation for such work.

References

Allen, J.; Hendler, J. and Tate, A. editors. *Readings in Planning.* Morgan-Kaufmann, San Mateo, CA, 1990.

Barrett, A. and Weld, D. Partial Order Planning. Technical report 92-05-01, Computer Science Dept., University of Washington, June, 1992.

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

Drummond, M. Refining and Extending the Procedural Net. In *Proc. IJCAI-85*, 1985.

Erol, K.; Nau, D.; and Subrahmanian, V. S. On the Complexity of Domain Independent Planning. In *Proc. AAAI-92*, 1992, pp. 381—387.

Erol, K.; Nau, D.; and Subrahmanian, V. S. When is planning decidable? In *Proc. First Internat. Conf. AI Planning Systems*, pp. 222–227, June 1992.

Erol, K.; Hendler, J.; and Nau, D. Semantics for Hierarchical Task Network Planning. Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, March 1994.

Erol, K.; Hendler, J.; and Nau, D. Complexity results for hierarchical task-network planning. To appear in *Annals of Mathematics and Artificial Intelligence* Also available as Technical report CS-TR-3240, UMIACS-TR-94-32, Computer Science Dept., University of Maryland, March 1994.

Erol, K.; Hendler, J.; and Nau, D. HTN Planning: Complexity and Expressivity To appear in *Proc. AAAI-94*, 1994.

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

Kambhampati, S. and Hendler, J. "A Validation Structure Based Theory of Plan Modification and Reuse" *Artificial Intelligence*, May, 1992.

Kambhampati, S. "On the utility of systematicity: understanding trade-offs between redundancy and commitment in partial-ordering planning," unpublished manuscript, Dec., 1992.

Lansky, A.L. Localized Event-Based Reasoning for Multiagent Domains. *Computational Intelligence Journal*, 1988.

McAllester, D. and Rosenblitt, D. Systematic nonlinear planning. In *Proc. AAAI-91*, 1991.

Minton, S.; Bresna, J. and Drummond, M. Commitment strategies in planning. In *Proc. IJCAI-91*, 1991.

Nilsson, N. *Principles of Artificial Intelligence*, Morgan-Kaufmann, CA. 1980.

Penberthy, J. and Weld, D. S. UCPOP: A Sound, Complete, Partial Order Planner for ADL *Proceedings of the Third International Conference on Knowledge Representation and Reasoning, October 1992*

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

Tate, A.; Hendler, J. and Drummond, D. AI planning: Systems and techniques. *AI Magazine*, (UMIACS-TR-90-21, CS-TR-2408):61–77, Summer 1990.

Tate, A. Generating Project Networks In Allen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 291—296.

Vere, S. A. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(3):246–247, 1983.

Wilkins, D. Domain-independent Planning: Representation and Plan Generation. In Allen, James; Hendler, James; and Tate, Austin, editors 1990, *Readings in Planning.* Morgan Kaufman. 319—335.

Wilkins, D. *Practical Planning: Extending the classical AI planning paradigm*, Morgan-Kaufmann, CA. 1988.

Yang, Q. Formalizing planning knowledge for hierarchical planning *Computational Intelligence* Vol.6., 12–24, 1990.