HTN Planning: Complexity and Expressivity*

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

Most practical work on AI planning systems during the last fifteen years has been based on hierarchical task network (HTN) decomposition, but until now, there has been very little analytical work on the properties of HTN planners. This paper describes how the complexity of HTN planning varies with various conditions on the task networks.

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 systems. There is much recent analysis of the properties of total- and partial-order planning systems using STRIPS-style planning operators—but STRIPS-style planning systems were developed more than 20 years ago, and most of the practical work on AI planning systems during the last fifteen years has been based on hierarchical task network (HTN) decomposition (e.g., NOAH(Sacerdoti, 1990), NONLIN(Tate, 1990), SIPE(Wilkins, 1988), and DEVISER(Vere, 1983)).

Until now, there has been very little analytical work on the properties of HTN planners. One of the primary obstacles impeding such work has been the lack of a clear theoretical framework explaining what a HTN planning system is, although two recent papers (Yang, 1990; Kambhampati *et al.*, 1992) have provided important first steps in that direction. A primary goal of our current work is to correctly define, analyze, and explicate features of the design of HTN planning systems.

Our work has progressed far enough that we can do complexity analyses of HTN planning similar to analyses which Erol *et al.* (1992) performed for planning with STRIPS-style operators. In particular, Table 1 shows how the complexity of telling whether a plan exists depends on the following factors: (1) restrictions on the existence and/or ordering of non-primitive tasks in task networks, (2) whether the tasks in task networks are required to be totally ordered, and (3) whether variables are allowed. From this table, we can draw the following conclusions:

1. HTN's are more expressive than STRIPS-style operators. This contradicts the idea, held by some researchers, that HTN's are just an "efficiency hack."
2. HTN planning is undecidable under even a very severe set of constraints. In particular, it is undecidable even if no variables are allowed, as long as there is the possibility that a task network can contain two non-primitive tasks without specifying the order in which they must be performed.
3. In general, what restrictions we put on the non-primitive tasks has a bigger effect on complexity than whether or not we allow variables, or require tasks to be totally ordered.
4. To achieve decidability, it is sufficient to place restrictions either on non-primitive tasks or on the ordering of tasks. If either restriction is removed individually, planning remains decidable, but removing both simultaneously makes planning undecidable.
5. If there are no restrictions on non-primitive tasks, then whether or not we require tasks to be totally ordered has a bigger effect (namely, decidability vs. undecidability) than whether or not we allow variables. But in the presence of restrictions on non-primitive tasks, whether or not we allow variables has a bigger effect than whether or not we require tasks to be totally ordered.

Basics of HTN Planning

Overview

To provide an intuitive feel for HTN planning, here is a deliberately oversimplified description. The "Details" section gives a more precise description.

The input to the planner consists of the following:

- An initial "task network" d representing the problem to be solved. A task network is a set of "tasks" representing things that need to be done. Each task is a task name along with a list of arguments, which may

*This work was supported in part by NSF Grant NSFD CDR-88003012 to the Institute for Systems Research, and NSF grant IRI9306580 and ONR grant N00014-91-J-1451 to the Computer Science Department.

Complexity of HTN Planning

Restrictions on non-primitive tasks	Must every HTN be totally ordered?	Are variables allowed?	
		no	yes
none	no	Undecidable	Undecidable[β]
	yes	in EXPTIME; PSPACE-hard	in DEXPTIME; EXPSPACE-hard
"regularity" [α]	doesn't matter	PSPACE-complete	EXPSPACE-complete
no non-primitive tasks	no	NP-complete	NP-complete
	yes	Polynomial time	NP-complete

[α] At most one non-primitive task, which must follow all primitive tasks.

[β] Even if the planning domain is fixed in advance.

be variables or constants. Some tasks are "primitive" (i.e., they can be performed directly), and others are "non-primitive" (i.e., the planner needs to figure out how to perform them). Task networks also include constraints on the tasks, which may restrict how some of the variables can be bound, the order in which the tasks are to be performed, etc.

- A set of "operators" Op telling the effects of each primitive task (action).
- A set of "methods" Me telling how to perform various non-primitive tasks. Each method is a pair $m = (t, d)$, where t is a task and d is a task network. It says one way to achieve t is to perform the tasks specified in the network d (provided that this can be done in a way that satisfies all the constraints).

Planning proceeds by starting with the the initial task network d, and doing the following steps repeatedly, until no non-primitive tasks are left: find a non-primitive task u in d and a method $m = (t, d')$ in M such that t unifies with u. Then modify d by "reducing" u (i.e., replace u with the tasks in d', and incorporate the constraints of d' into d). Once no non-primitive tasks are left in d, the next problem is to find a totally-ordered ground instantiation σ of d that satisfies all of the constraints. If this can be done, then σ is a successful plan for the original problem.

In practice, HTN planning also has several other aspects. In particular, functions are often provided which can "debug" partially reduced task networks to eliminate potential problems. These "critic" functions are used to handle ordering constraints, resource limits, and to provide domain-specific guidance. The formalization described in (Erol *et al.*, 1994a) explains critics and the relationship between these and the constraints described above. For the purposes of this paper, the critics do not affect worst-case behavior, and thus we will omit this detail.

Details

Our language $\mathcal{L}$ for HTN planning is a first-order language with some extensions. The representations of the world and the actions in HTN planning is very similar to those of STRIPS-style planning. Thus, $\mathcal{L}$ contains a set C of constant symbols that represent the objects, and a set P of predicate symbols that represent the relations among the objects. $\mathcal{L}$ also contains a set F of primitive task symbols which represent the actions. We use constructs called *operators* to associate effects to primitive task symbols. We define a plan as a sequence of ground primitive tasks, and we designate the *initial state* of the world by a list of ground atoms.

The fundamental difference between STRIPS-style planning[1] and HTN planning is the representation of "desired change" in the world. HTN planning replaces STRIPS-style "goals" with tasks and task networks (which we later show are more powerful). There are three types of tasks:

- *Goal tasks*, like goals in STRIPS, are properties we wish to make true in the world (for example, having a new house).
- *Primitive tasks* are the 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.). Compound tasks allows us to represent "desired changes" that can not be represented as a single goal task or primitive task. As an 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, the compound task of making a round trip to New York cannot easily be expressed as a single goal task, because the initial and final states would be the same.

Formally, the vocabulary of HTN language $\mathcal{L}$ is a tuple $\langle V, C, P, F, T, N \rangle$, where $V = \{v_1, v_2, \ldots\}$ 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, T is a finite set of *compound* task symbols, and $N = \{n_1, n_2, \ldots\}$ is

[1] We use the term "STRIPS-style" planning to refer to any planner (either total- or partial-order) in which the planning operators are STRIPS-style operators (i.e., operators consisting of three lists of atoms: a precondition list, an add list, and a delete list). These atoms are normally assumed to contain no function symbols.

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

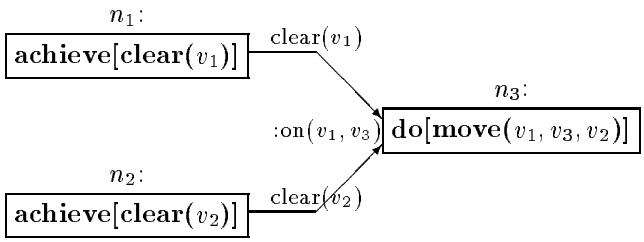


Figure 1: A task network, and its graphical representation.

an infinite set of symbols used for labeling tasks. If $x_1, \ldots, x_k$ are terms, then a *primitive task* has the form $do(f(x_1, \ldots, x_k))$, where $f \in F$; a *goal task* has the form $achieve(l)$, where l is a literal; and a *compound task* has the form $perform[t(x_1, \ldots, x_k)]$, where $t \in T$. We refer to goal tasks and compound tasks as non-primitive tasks.

Tasks are connected together in HTN planning via the use of task networks,[2] which are collections of tasks and constraints on those tasks. Formally, a *task network* has the form $((n_1 : \alpha_1), \ldots, (n_m : \alpha_m), \phi)$, where each α_i is a task labeled with n_i, and 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 $n, n' \in N$, $v, v' \in V$, l is a literal, and $c \in C$. $n \prec n'$ means that the task labeled with n precedes the one labeled with n'; (n, l), (l, n) and (n, l, n') mean that l needs to be true immediately after n, immediately before n, and between n and n', respectively. Both negation and disjunction are allowed in the constraint formula.

As an example, Fig. 1 shows a blocks-world task network and its graphical representation. In this task network there are three tasks: clearing v_1, clearing v_2, and moving v_1 to v_2. The task network also includes the constraints 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.

To specify how actions change the world, we use *operators* of the form $(f(v_1, \ldots, v_k), l_1, \ldots, l_m)$, where f is a primitive task symbol, $v_1, \ldots, v_k$ are variable symbols, and $l_1, \ldots, l_m$ are literals, denoting the primitive task's effects (which are also called postconditions). Our HTN operators do not contain STRIPS-style preconditions; preconditions are realized as goal tasks in task networks (as in Fig. 1).

It is clear how to achieve a primitive task: execute the corresponding action. But for non-primitive tasks, we need to tell our planner how to achieve them, and we do this using constructs called *methods*.

A *method* is a pair (α, d) where α is a non-primitive task, and d is a task network. It states that one way of achieving the task α is to achieve the task network d, i.e to achieve all the subtasks in the task network without violating the constraint formula of the task network. For example, a blocks-world method for achieving $on(v_1, v_2)$ would look like $(achieve(on(v_1, v_2)), d)$, where d is the task network in Fig. 1. An empty plan would achieve a goal task when the goal is already true. Thus, for each goal task, we (implicitly) have a method $(achieve(l), ((n : do(t))(l, n)))$ which contains only one dummy primitive task t with no effects, and the constraint that the goal l is true immediately before t.

Planning Domains and Problems

A planning domain is a pair $\mathcal{D} = \langle Op, Me \rangle$, where Op is a set of operators, and Me is a set of methods.

A *planning problem* is a triple $\mathbf{P} = \langle d, I, \mathcal{D} \rangle$, where $\mathcal{D}$ is a planning domain, I is the initial state, and d is the task network we need to plan for. The language of $\mathbf{P}$ is the HTN language $\mathcal{L}$ generated by the constant, predicate, and task symbols appearing in $\mathbf{P}$, along with an infinite set of variables and an infinite set of node labels. Thus, the set of constants, predicates and tasks are all part of the input.

$\mathbf{P}$ is *primitive* if the task network d contains only primitive tasks. $\mathbf{P}$ is *regular* if all the task networks in the methods and d contain at most one non-primitive task, and that non-primitive task is ordered to occur as either the first or the last task. $\mathbf{P}$ is *propositional* if no variables are allowed. $\mathbf{P}$ is *totally ordered* if all the tasks in any task network are totally ordered.

PLAN EXISTENCE is the following problem: given $\mathbf{P} = \langle d, I, \mathcal{D} \rangle$, is there a plan that solves $\mathbf{P}$?

The problem of finding an *optimal* (i.e., shortest-length) plan that solves $\mathbf{P}$ is at least as difficult as the problem of determining whether or not a plan exists. In an analysis of STRIPS-style planning, Erol *et al.* (1992) analyzed this problem by transforming it into a decision problem (which we called PLAN LENGTH) according to the usual complexity-theoretic technique of asking whether, for some input integer k, there exists a successful plan of length k or less.

This paper does not address the plan optimality problem, for two reasons. First, HTN planners have usually not worried about optimality because it is so difficult to verify (in many cases, optimality cannot be guaranteed by method decomposition). Second, Erol *et al.* (1992) found that for STRIPS-style planning, in some cases the complexity of PLAN LENGTH was misleadingly low. In particular, PLAN LENGTH was NEXPTIME-complete even in cases where the plan optimality problem was much harder, because the input

[2] These are also called "procedural nets" in some of the literature (Sacerdoti, 1990; Drummond, 1985).

to PLAN LENGTH includes the integer k encoded in binary, which confines the planner to plans of length at most exponential in the length of the input.

Operational Semantics

In this section, we give a syntactic characterization of the set of solutions for a given HTN-planning problem. Description of an equivalent model-theoretic semantics appear in (Erol *et al.*, 1994a).

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.

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'. Then 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 $d'\theta$, and incorporating $d'\theta$'s constraint formula into the constraint formula of d. We denote the set of reductions of d by $red(d, I, \mathcal{D})$. Reductions formalize the notion of *task decomposition*. For a precise definition of completions and reductions, the reader is referred to (Erol *et al.*, 1994a).

Here are the two inference 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 $\sigma \in sol(d, I, \mathcal{D})$.

Rule R1 says that the set of plans that achieve a primitive task network consists of the completions of the task network; Rule R2 says that if d' is a reduction of d, then any plan that achieves d' also achieves d.

Now, we need to define the set of plans that can be derived using those two inference rules. Let us define a function $sol(d, I, \mathcal{D})$ as follows:

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

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. In (Erol *et al.*, 1994a), it is proved that $sol()$ is indeed the set of solutions, and that the inference rules **R1**, **R2** are sound and complete.

Results

Decidability

It is easy to show that we can simulate context-free grammars within HTN planning. More interesting is the fact that we can simulate any two context-free grammars, and with the help of task interleavings and constraints, we can check whether these two grammars have a common string in the languages they generate. Whether the intersection of the languages of two context-free grammars is non-empty is a semi-decidable problem (Hopcroft *et al.*, 1979). Thus:[3]

Theorem 1 PLAN EXISTENCE *is strictly semi-decidable, even if* **P** *is restricted to be propositional, to have at most two tasks in any task network, and to be totally ordered (except for the input task network).*

One way to make PLAN EXISTENCE decidable is to restrict the methods to be acyclic. In that case, any task can be expanded up to a finite depth, and thus the problem becomes decidable. To this end, we define a *k-level-mapping* to be a function $level()$ from ground instances of tasks to the set $\{0, \ldots, k\}$, such that whenever we have a method that can expand a ground task t to a task network containing a ground task t', $level(t) > level(t')$. Furthermore, $level(t)$ must be 0 for every primitive task t.

Intuitively, $level()$ assigns levels to each ground task, and makes sure that tasks can be expanded into only lower level tasks, establishing an acyclic hierarchy. In this case, any task can be expanded to a depth of at most k. Therefore,

Theorem 2 PLAN EXISTENCE *is decidable if* **P** *has a k-level-mapping for some finite integer k.*

Another way to make PLAN EXISTENCE decidable is to restrict the interactions among the tasks. Restricting the task networks to be totally ordered limits the interactions that can occur between tasks. Tasks need to be achieved serially, one after the other; interleaving subtasks for different tasks is not possible. Thus interactions between the tasks are limited to the input and output state of the tasks, and the "protection intervals", i.e the literals that need to be preserved.

Under the above conditions, we can create a table with an entry for each task, input/output state pair, and set of protected literals, that tells whether it is possible to achieve that task under those conditions. Using dynamic programming techniques we can compute the entries in the table in DOUBLE-EXPTIME, or in EXPTIME if the problem is further restricted to be propositional. It is easy to show that STRIPS-style planning can be modeled using HTN's that satisfy these conditions, so we can use the complexity results on STRIPS-style planning in (Erol *et al.*, 1992) to establish a lower bound on the complexity of HTN planning. Thus:

Theorem 3 PLAN EXISTENCE *is* EXPSPACE*-hard and in* DOUBLE-EXPTIME *if* **P** *is restricted to be totally ordered.* PLAN EXISTENCE *is* PSPACE*-hard and in* EXPTIME *if* **P** *is further restricted to be propositional.*

If we restrict our planning problem to be regular, then there will be at most one non-primitive task in any task network (both the initial input task network,

[3] All proofs appear in (Erol *et al.*, 1994b).

and those we obtain by expansions). Thus, subtasks in the expansions of different tasks cannot be interleaved, which is similar to what happens in Theorem 3. But in Theorem 3, there could be several non-primitive tasks in a task network, and we needed to keep track of all of them (which is why we used the table). If the planning problem is regular, we only need to keep track of a single non-primitive task, its input/final states, and the protected literals. Since the size of a state is at most exponential, the problem can be solved in exponential space. But even with regularity and several other restrictions, it is still possible to reduce an EXPSPACE-complete STRIPS-style planning problem (described in (Erol *et al.*, 1992)) to the HTN framework. Thus:

Theorem 4 PLAN EXISTENCE *is* EXPSPACE-*complete if* **P** *is restricted to be regular. It is still* EXPSPACE-*complete if* **P** *is further restricted to be totally ordered, with at most one non-primitive task symbol in the planning language, and all task networks containing at most two tasks.*

When we further restrict our problem to be propositional, the complexity goes down one level:

Theorem 5 PLAN EXISTENCE *is* PSPACE-*complete if* **P** *is restricted to be regular and propositional. It is still* PSPACE-*complete if* **P** *is further restricted to be totally ordered, with at most one non-primitive task symbol in the planning language, and all task networks containing at most two tasks.*

Suppose a planning problem is primitive, and either propositional or totally ordered. Then the problem's membership in NP is easy to see: once we nondeterministically guess a total ordering and variable binding, we can check whether the constraint formula on the task network is satisfied in polynomial time. Furthermore, unless we require the planning problem to be both totally ordered and propositional, our constraint language enables us to represent the satisfiability problem, and thus we get NP-hardness. Hence:

Theorem 6 PLAN EXISTENCE *is* NP-*complete if* **P** *is restricted to be primitive, or primitive and totally ordered, or primitive and propositional. However,* PLAN EXISTENCE *can be solved in polynomial time if* **P** *is restricted to be primitive, totally ordered, and propositional.*

Expressivity

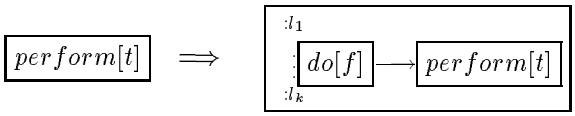
It has been informally observed that HTN approaches do not need to completely specify the conditions that each action affects, while the STRIPS-style "state-based" plan structures typically require complete specification of intermediate states. Thus, in describing the relationships between actions, it has been argued that HTN approaches are more appropriate. Lansky (1988), for example, makes this argument and claims it is largely responsible for the the more general use of HTNs over STRIPS-style systems in planning practice.

Despite such claims, it has never been demonstrated that HTNs can encode situations which STRIPS-style planning operators cannot, because the lack of a formalism for HTN planning has left it unclear what can be expressed with HTNs. Using the formalism in this paper, we can directly compare the expressive power of HTN and STRIPS-style planning operators.

When we compare HTNs and STRIPS, we observe that the HTN approach provides all the concepts (states, actions, goals) that STRIPS has. In fact, given a domain encoded as a set of STRIPS operators, we can transform it to an HTN planning domain, in low-order polynomial time. A straightforward transformation would be to declare one primitive task symbol for each STRIPS operator, and for every effect of each operator, to declare a method similar to the one in Fig. 1. Each such method contains the preconditions of the operator as goal tasks, and also the primitive task corresponding to the operator itself.

Below is a more instructive transformation, which demonstrates that the relationship between STRIPS-style planning and HTN planning is analogous to the relationship between right linear (regular) grammars and context-free grammars. We summarize the transformation below; for details see (Erol *et al.*, 1994b).

In this transformation, the HTN representation uses the same constants and predicates used in the STRIPS representation. For each STRIPS operator o, we declare a primitive task f with the same effects as o. We also use a dummy primitive task f_d with no effects. We declare a single compound task symbol t. For each primitive task f, we construct a method of the form



where $l_1, \ldots, l_k$ are the preconditions of the action associated with f. We declare one last method $\boxed{perform[t]} \Rightarrow \boxed{do[f_d]}$. Note that t can be expanded to any sequence of actions ending with f_d, provided that the preconditions of each action are satisfied. The input task network has the form $[(n : perform[t]), (n, G_1) \wedge \ldots \wedge (n, G_m)]$ where $G_1, \ldots, G_m$ are the STRIPS-style goals we want to achieve Note that the transformation produces regular HTN problems, which has exactly the same complexity as STRIPS-style planning. Thus, just as restricting context-free grammars to be right linear produces regular sets, restricting HTN methods to be regular produces STRIPS-style planning.

HTNs can express situations impossible to express using unmodified STRIPS operators. Intuitively, this is because STRIPS lacks the concept of compound tasks, and its notion of goals is limited. It does not provide means for declaring goals/constraints on the intermediate states as HTNs do. Furthermore, in contrast to STRIPS, HTNs provide a rich constraint language that can express many types of interactions.

More formally, from Theorem 1, HTN planning with no function symbols (and thus only finitely many ground terms) is semi-decidable. Even if we require the domain description $\mathcal{D}$ to be fixed in advance (i.e., not part of the input), there are HTN planning domains for which planning is semi-decidable.[4] However, with no function symbols, STRIPS-style planning is decidable, regardless of whether or not the planning domain[5] is fixed in advance (Erol *et al.*, 1992). Thus:

Theorem 7 *There exists* HTN *planning domains that can not be represented by any finite number of* STRIPS*-style operators.* [6]

Another way of comparing expressive power of two languages is based on model-theoretic semantics, which we do in (Erol *et al.*, 1994a).

The power of HTN planning comes from two things: (1) allowing multiple tasks and arbitrary constraint formulas in task networks, (2) compound tasks. Allowing multiple tasks and arbitrary formulae provides flexibility—but if all tasks were either primitive or goal (STRIPS-style) tasks, these could probably be expressed with STRIPS-stye operators (albeit clumsily and using an exponential number of operators/predicates). Compound tasks provide an abstract representation for sets of primitive task networks, similar to the way non-terminal symbols provide an abstract representation for sets of strings in context-free grammars.

Conclusion

Our results show that handling interactions among non-primitive tasks is the most difficult part of HTN planning. In particular, if subtasks in the expansions for different tasks can be interleaved, then planning is undecidable, even if no variables are allowed.

We have investigated several conditions on the planning problem, such as restricting task-networks to contain a single non-primitive task or to be totally ordered. Those restrictions reduced the complexity significantly, because they limited the interactions among tasks.

Our comparison of the complexity of HTN planning and STRIPS-style planning demonstrates that HTN planners can represent a broader and more complex set of planning problems and planning domains. The transformations from HTN planning problems to STRIPS-style planning problems have revealed that STRIPS-style planning is a special case of HTN planning, and that the relation between them is analogous to the relation between context-free languages and regular languages.

[4](Erol *et al.*, 1994b) includes several complexity results similar to those in this paper, for the case when $\mathcal{D}$ is fixed.

[5]Since STRIPS-style planning does not include methods, a STRIPS-style planning domain is simply a set of operators.

[6]In proving this theorem, we use the standard assumption that the STRIPS operators do not contain function symbols, nor do the HTN operators.

Acknowledgement

We thank R. Kambhampati and A. Barrett for their insightful comments.

References

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. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* to appear. A more detailed version is available as Tech. Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96, University of Maryland, College Park, MD, 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.

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) 1971.

Hopcroft and Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company Inc., California, 1979.

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

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

Sacerdoti, E. D. The nonlinear Nature of Plans In Allen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 162—170.

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

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