A Study of Process Languages for Planning Tasks

Stephen Lee-Urban, Héctor Muñoz-Avila

Department of Computer Science & Engineering, Lehigh University, Bethlehem, PA 18015-3084 USA. {sml3, hem4}@lehigh.edu

Abstract

We present the semantics of processes for planning tasks. We target Task-Method-Knowledge (TMK) process models, a widely used formalization of processes. These semantics are formalized by proving that Task-Method-Knowledge (TMK) process models can be translated into equivalent Hierarchical Task Networks (HTNs). We construct a Turing-computable translation function that maps TMK process model constructs to equivalent HTN constructs. We leverage well-known results from HTN planning to define soundness, completeness and complexity results for TMKs.

Introduction

Processes are an important family of knowledge constructs. Loosely speaking, a *process* is the means by which tasks are accomplished via a series of actions or operations. Process modeling is important when reasoning by concept reuse and modification; without such a model, reasoning takes place at the level of actions, such as STRIPS operators, and knowledge transfer to new problems is difficult. Process models ease knowledge transfer by capturing how to achieve a goal. This yields a flexible representation for problem solving.

Processes have been used in a wide range of applications. For example, they have been used to: represent web services integration (Curbera *et al*., 2002), build adaptive agents (Ulam *et al*,. 2005), build introspective agents that use models to identify failures in their reasoning (Murdock, 2001), and build a testbed for machine learning systems (Molineaux & Aha, 2005). For our analysis, we use the process language called *Task-Method-Knowledge process models* (Murdock, 2001), which was also used in the latter three applications mentioned above.

One drawback of Task-Method-Knowledge process models (TMKs) is that they lack clear semantics, in particular when viewed as a language for expressing planning problems as used in REM (Murdock, 2001) and TIELT (Molineaux & Aha, 2005). In this paper we define formal semantics for TMKs by demonstrating their equivalence with Hierarchical Task Networks (HTNs). This will allow us to state soundness and completeness results for planning with TMKs, show that planning with TMKs is NP-complete, and prove that they are strictly more expressive than STRIPS plans. Furthermore, we prove that a Turing-computable translation function exists from TMKs to HTNs that runs in polynomial time.

The significance of these results is threefold. First, the results imply that HTN reasoning capabilities that have clear semantics can be applied to the growing body of TMKs knowledge. Second, several advances involving HTNs, including results on their applicability (Wilkins, 1988) and learning them (Choi & Langley, 2005; Ilghami *et al*., 2005), can be applied to a wide range of real-world applications for which TMKs knowledge is readily available. Third, TMKs adopt common characteristics of process languages, including grouping of simple activities into more complex activities, iteration (e.g., *while* loops), variable assignment, and value return. Thus, our results should also apply to other process languages.

Hierarchical-Task-Network Planning

HTN planning is a form of planning that reasons at the level of high-level tasks instead of lower-level actions (Erol *et al*., 1994). In HTN planning, high-level tasks are repeatedly decomposed into simpler ones until all tasks have been reduced to primitive actions.

The main knowledge artifacts in HTN planning are called *methods*. A method encodes how to achieve a compound task. Methods consists of three elements: (1) the task being achieved, called the *head* of the method, (2) the set of *preconditions* indicating the conditions that must be fulfilled for the method to be applicable, and (3) the *subtasks* needed to achieve the head. The second set of knowledge artifacts are the *operators*. Operators in HTN planning have the same purpose as in STRIPS planning: they represent action schemes. Like STRIPS operators, an HTN operator consists of the primitive task it achieves and its effects, indicating how the world changes when it is applied. However, HTN operators have no preconditions because applicability conditions are determined in the methods.

Task-Method-Knowledge Planning

TMK planning is another form of planning that, like HTN planning, reasons at the level of high-level tasks instead of low-level actions. TMKs capture tasks (what an agent does), methods (how the agent works), and knowledge (the information used by the agent). Tasks are accomplished by methods, which are in turn further decomposed into lower level tasks as specified by the methods. A hierarchy is consequentially created where the leaves of the resulting TMK model tree are primitive tasks (not decomposable) that explicitly specify their effects, and the non-primitive

tasks are the internal nodes of the tree (Murdock, 2001). The set of plans representable TMK models is a superset of the set of plans representable as HTNs.

As with HTNs, *methods* are behavioral elements that encode how a piece of computation works; the overall function of a method is encoded in the task it addresses. Methods specify the means of accomplishing a task, and the applicability of a selected method to a particular task is determined by the method's preconditions and the current situation. A TMK method has three elements: *preconditions* that must be true in order to perform the method, *postconditions* that represent the consequences of performing it, and a reference to the first transition in a state-transition machine that implements the method's behavior. A method can be viewed as a state machine, which is convenient for describing the translation of statements (e.g., while, for, if). TMK operators have the same form as in STRIPS planning.

Translating TMKs into HTNs: A Sketch

Our translator selects each input TMK method and calls a recursive procedure that traverses its state-transition machine implementation. This state machine can be viewed as a sequence of statements s_1, .., s_n, where each statement s_i can be a complex statement consisting of its own statements. Our current implementation translates TMKs from TIELT (Molineaux, & Aha, 2005) into HTNs for the SHOP HTN planner (Nau *et al*., 1999).

The procedure HTNs() translates a list of TMK methods M_T into an equivalent collection M_T of HTN methods (Fig. 1). For every TMK method m_T we call the procedure toHTNs(), with the body of m_T and a dummy task that can always be accomplished (line 1-2). This procedure returns the collection M of HTN methods translating the body and a task t_{body} that is accomplished by methods in M (line 3). We construct a new method m, that decomposes t into t_{body} (line 4). All these are collected and returned (lines 5-6).

The procedure toHTNs() is also shown in Figure 2. It receives as input a sequence of TMKs statements E that must be executed before a task t_{next} (initially called with a dummy task). This procedure translates a finite state machine's statements in the reverse order (s_n, ..., s_1) in which they appear in the machine and each statement s_k is translated from the most to the least deeply nested. If E has only one statement s1 that has no body, we create a new task *nt* accomplished by a new method *m* that decomposes into the translation of that statement (lines 5-7). The task t_{next} is appended as the last subtask of the method (line 8). The pair (m,nt) is then returned (line 9). If E has only one statement s1 that has a body (i.e., one or more nested statements), then we return the result of a recursive call (lines 3 and 4). If E has more than one statement, these are translated in reverse order (lines 10-14). A recursive call is in line 12. We update the task t_{next} for the next iteration to be the task t returned by the previous recursive call (line 13). During each iteration, the resulting HTN methods are collected in M_H (line 14). Finally the pair (M_H,t) is returned (line 15).

Table 1 informally summarizes how the individual statements are translated in the algorithm toHTNs().

More compact translations are possible if HTN planners support expressions defined in the planning language PDDL (Long & Fox, 2002). PDDL is a STRIPS-based planning language, a subset of which is used in the annual AI planning competition. However, the mappings described here are the most direct and suffice to illustrate the equivalence between TMKs and HTNs.

Procedure HTNs(M_T)
Input: M_T is a list of TMKs methods
Output: A collection M_H of HTN methods translating M_T
1 $M_H \leftarrow \{\}$
2 **for** every TMKs method m_T =(t,body) in M_T **do**
3 (M,t_{body}) ← toHTNs(body,dummyTask())
4 m ← newMethod(t, <t_{body}>)
5 $M_H \leftarrow M_H \cup M \cup \{ m \}$
6 **return** M_H

Procedure toHTNs(E,t_{next})
Input: E is a sequence of TMKs statements that must be executed before t_{next}
Output: A collection M_H of HTN methods translating E and the task t that achieve the methods in M_H
1 **Let** E = (s_1, .., s_n)
2 **if** (n = 1) **then**
3 **if** (s_1 is complex) **then**
4 **return** toHTNs (s_1,t_{next})
5 **else**
6 nt ← newTask()
7 m ← newMethod(nt, translateAtomicStatement(s_1))
8 appendMethod(m, t_{next})
9 **return** ({m}, nt)
10 **else**
11 **for** k ← n to 1 do
12 (M,t) ← toHTNs(s_n, t_{next})
13 t_{next} ← t
14 $M_H \leftarrow M_H \cup M$
15 **return** (M_H,t)

Figure 1: Pseudo-code of the TMKs Translation Algorithm

TMKs	HTNs
Return (values from functions)	Use an unbound variable as a parameter in the caller's invocation; set same variable in called method's preconditions
If-then-else	Use HTN method syntax
While (iteration)	Use recursion
For (iteration)	Change to while, use recursion
Tasks with preconditions, effects	Add preconditions to methods and the effects as new tasks
Call	Subtask
Set (variable assignment)	Split into a new method and pass the variables by value

Table 1: Mapping from TMKs to HTNs

Semantics and Complexity of TMKs Planning

According to Erol *et al.* (1994) no well-established definition of planning language expressivity exists; expressivity has instead been defined using model-theoretic semantics, operational semantics, and the computational complexity of problems representable by the planning language.

One way to demonstrate the expressive equivalence of HTNs and TMKs is a presentation like that described in (Erol *et al.*, 1995), which states that the expressivity of two languages can be compared by demonstrating that a polynomial or Turing computable transformation exists between them. We presented a sketch of this transformation from TMKs to HTNs in the previous section. This sketch allows us to assert the following:

> **Theorem:** There exists a Turing-computable function ψ from the set of TMK planning problems to the set of HTN planning problems such that, for any TMK planning problem $\mathbf{P}$ and any plan σ, σ solves $\mathbf{P}$ iff σ solves the HTN planning problem $\psi(\mathbf{P}) = \mathbf{P_H}$, under the assumption that $\mathbf{P}$ and $\mathbf{P_H}$ have the same set of operators.

We define a TMK planning problem as a triple $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, where $\mathbf{T} = (M_T, O)$ is a TMK process model, I is the initial state, and d is the task network for which a plan is desired. Therefore, the translation function ψ defines a translation from M_T into M_H because d, I, and O are the same for both $\mathbf{P}$ and $\mathbf{P_H}$.

Given the existence of a transformation, both $\mathbf{P}$ and σ can apply to HTNs and therefore lend their well-defined semantics to TMKs. In particular:

- **TMK Planning is Sound and Complete**. This means that if a planning problem is solvable, a correct solution plan will be found. Informally, given a TMK planning problem $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, a plan that recursively achieves all tasks in d is called a correct plan of the planning problem (Nau *et al*., 1999). Given a TMK problem $\mathbf{P} = \langle d, I, \mathbf{T} \rangle$, where $\mathbf{T} = (M_T, O)$, the translation $\mathbf{H} = (\psi(M_T), O)$ defines an HTN planning domain. The solution plan for $\mathbf{P_H} = \langle d, I, \mathbf{H} \rangle$ is a sequence of actions, each an instance of an operator in O. This same sequence is also a solution for $\mathbf{P}$.
- **TMKs are more expressive than STRIPS**. This means that, for every Turing-computable function ψ from the set of TMK planning problem instances to the set of STRIPS planning problem instances, there is a TMK planning problem instance $\mathbf{P}$, for which either $\psi(\mathbf{P})$ is not defined or, if it is defined, σ does not solve $\psi(\mathbf{P})$. This result follows from HTNs being provably more expressive than STRIPS (Erol *et al*., 1994) and because TMKs can always be represented as equivalent HTNs.
- **TMK planning is NP-Complete**.

Conclusions and Future Work

In this paper we described formal semantics for Task-Method-Knowledge process models by constructing a Turing-computable translation function that maps TMKs into equivalent HTN constructs. This allowed us to state soundness, completeness, and complexity results for planning with TMK process models, and prove that TMK planning is strictly more expressive than STRIPS planning. Furthermore, we proved that this translation function runs in polynomial time.

In our future work, we wish to conduct experiments to confirm the theoretical results. We wish to empirically show that the overhead of plan generation running time with a translated domain is linearly correlated to an equivalent domain natively encoded in an HTN. We will explore solving project planning problems translated from TMKs into HTNs. We also wish to explore our conjecture that the translation works for other process languages.

References

Choi, D., & Langley, P. 2005. Learning teleoreactive logic programs from problem solving. *Proceedings of the Fifteenth International Conference on Inductive Logic Programming*. Bonn, Germany: Springer.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., Weerawarana, S. 2002. Business Process Execution Language for Web Services, Version 1.0. *IBM technical report*.

Erol, K., Hendler, J., & Nau, D. 1994. HTN planning: Complexity and Expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (AAAI-94). AAAI Press

Erol, K., Hendler, J., & Nau, D. 1995. *Complexity results for HTN planning*. Annals of Mathematics and Artificial Intelligence. Kluwer.

Ilghami, O., Muñoz-Avila, H., Nau, D. S., and Aha, D. W. 2005. Learning approximate preconditions for methods in hierarchical plans. *Proceedings of the International Conference on Machine Learning* (ICML-05)

Fox, M., & Long, D. 2003. pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research.*.

Molineaux, M., & Aha, D.W. 2005. TIELT: A Testbed for Gaming Environments . In: Proceedings of the Twentieth National Conference on Artificial Intelligence. Pittsburgh, PA: AAAI Press.

Murdock, J. W. 2001. *Self-Improvement through Self-Understanding: Model-Based Reflection for Agent Adaptation*. Dissertation Georgia Institute of Technology.

Ulam, P., Goel, A., Jones, J. & Murdock, J.W. 2005. Using Model-Based Reflection to Guide Reinforcement Learning. *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation and Learning in Computer Games*. Edinburgh, UK.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc.