Expressivity of STRIPS-like and HTN-like Planning

Marián Lekavý and Pavol Návrat

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia,
lekavy@fiit.stuba.sk, navrat@fiit.stuba.sk

Abstract. It is widely believed, that the expressivity of STRIPS and STRIPS-like planning based on actions is generally lower than the expressivity of Hierarchical Task Network (HTN) and HTN-like planning, based on hierarchical decomposition. This would mean that a HTN-like planner can generally solve more domains than a STRIPS-like planner with the same extensions. In this paper, we show that both approaches, as they are practically used, are identically expressive and can solve all domains solvable by a Turing machine with finite tape (i.e. solvable by a common computer).

1 Introduction

Two most known and used approaches to domain-independent symbolical planning are STRIPS-like (based on operators) and HTN-like (based on hierarchical decomposition) planning. STRIPS-like planning is older and based on creating a plan as a chain of actions, while each of these actions has its precondition and results. HTN-like planning is based on hierarchical decomposition - the planner starts from an initial task and decomposes it into more primitive tasks according to decomposition information given to the planner. The primitive non-decomposable tasks form the final plan.

HTN was created as an extension to the "classical" STRIPS-like planning, allowing the planner to use additional information about the hierarchical decomposition.

As hierarchical decomposition is an extension of operator-based planning, the question of expressivity arose: Is the expressivity of HTN-like planning larger than the expressivity of STRIPS-like planning? Can HTN-like planning solve more domains than STRIPS-like planning?

This question was answered positively in the past[2]. The proof was, however, based on the assumption, that the HTN planner can use an infinite set of symbols to mark the tasks. This can be true for the theoretical HTN model, but this assumption cannot be fulfilled by any practically usable planner, implemented on a common computer.

This paper shows, that HTN-like planning expressivity is identical to STRIPS-like planning expressivity, under the assumption that we use any restriction,

which makes the HTN-planning process decidable. This assumption is not very restricting, as every implemetnation of HTN-like planning uses this kind of restriction to end the computation in finite time (even if unsuccesfull).

In this section, we will provide a brief overview of STRIPS-like and HTN-like planning. Section 2 provides discussion on expressivity of both approaches. In section 3, we show that STRIPS can emulate the finite tape Turing machine with the same time complexity. Finally, section 4 describes a conversion of HTN-like domains to STRIPS-like domains.

1.1 STRIPS

The basic principle of STRIPS[3] and STRIPS-like planning is finding a sequence of actions, which will modify the initial state of the world into the final state of the world. The state of the world is expressed in the form of a set of literals. The planner adds actions incrementally to the plan, trying to create the correct transformation from initial to the final state.

The STRIPS planning is based on operators in the form $Op = (pre, del, add)$, where *pre* is a precondition, which has to be valid immediately before the operator is applied, *add* / *del* are the sets of literals added / deleted to / from the world state after the operator ends. An executed operator (added to a plan) is called action.

Since STRIPS (35 years ago), there are plenty different planners based on the idea STRIPS used. Most planners are not limited to the basic STRIPS formalism, but have extensions like resources, parallel execution, timed actions, sensory actions and coordination actions, conditional and contingent actions etc.

The basic algorithm of STRIPS-like planning is based on sequential adding actions to a plan. When starting from the initial state, we speak about forward chaining, when starting from the final state, it is backchaining and if we add actions on an arbitrary place of the plan, then it is plan-space search. We construct the plan based only on the knowledge of the operators' precondition and effects and the current world state. No additional information is provided to the planner, which is the main difference to the HTN-like approaches, described in the next section.

1.2 Hierarchical Task Network (HTN)

The HTN[7] and HTN-like approaches are based on hand-made hierarchical decomposition of the problem domain. The planner is provided with domain knowledge, expressed as the possible decompositions of tasks into subtasks. Tasks can be primitive (directly executable) and non-primitive. Non-primitive tasks have to be decomposed into other tasks. Each non-primitive task has one or more lists of tasks, it can be decomposed into. This list of tasks, together with other restrictions (like precedence of tasks, variable binding or mutual exclusion) is called task network.

The creation of a plan starts with one or more initial tasks, which are decomposed into simpler tasks, until all tasks are decomposed into primitive tasks.

If the decomposition is not possible (e.g. because of colliding restrictions), the planner backtracks and creates a different decomposition.

The decomposition may be fully ordered, but there are also planners which allow interleaving of subtasks of different tasks (e.g. SHOP2[6]).

HTN can also be seen as an augmentation of action-based planning by a grammar, pruning the plan space[4].

The basic HTN algorithm is as follows:

```
1. Insert the initial tasks into the plan.
2. If the plan contains only primitive tasks, success.
3. Choose one non-primitive task from the plan.
4. Replace the chosen task by its subtasks according to some task network.
5. Resolve interactions and conflicts in the plan.
   If not possible, backtrack.
6. Continue with step 2.
```

Similar to STRIPS-like planning, HTN-like planners also introduce a variety of extensions like resources handling, parallel execution, timed actions, sensoric and coordination actions. HTN-like planners have an important role in planning for multi-agent systems. Planners like STEAM[8] or PGP/GPGP[5] introduce coordination and negotiation mechanisms, allowing multiple agents to participate in fulfilling common tasks. While STEAM introduces team tasks, which are decomposed into tasks for individual agents, in PGP, each agent has its own initial task that has to be fulfilled and parts of the decomposition tree can overlap for different agents (Figure 1).

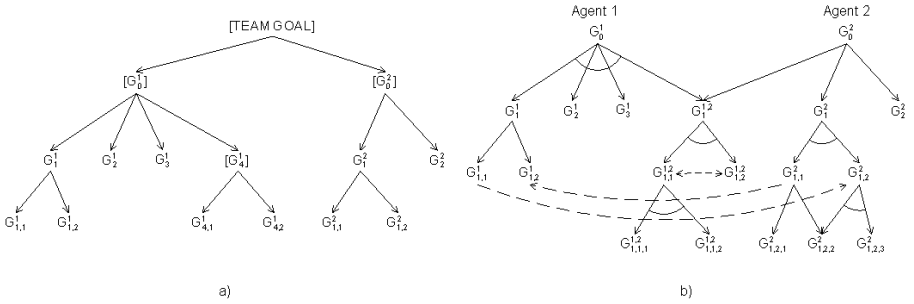

Fig. 1. An example of decomposition tree for STEAM (a) and PGP (b).

2 Expressivity

It is widely believed, that HTN-like planning expressivity is larger than the expressivity of STRIPS-like planning. By expressivity, we mean the set of planning domains the planning system is able to solve.

There is a proof[2], based on converting the planning problem to grammars and then showing, that STRIPS-like planning corresponds with regular grammars and HTN-like planning corresponds with context-free grammars. This way, it is shown that the HTN-like planning covers a wider class of possible planning domains.

The main difference, causing this result, is the fact, that the theoretical model of HTN uses an infinite set of symbols for marking the tasks, thus having the possibility of an infinite plan-space even in a simple domain. On the other hand, STRIPS and STRIPS-like planning have a finite plan-space, if not using some extensions (usually considered to be non-standard). Some STRIPS-like planners (like the FHP[9]) extend the basic formalism by including function symbols, allowing them to express undecidable problems.

It is clear, that the theoretical HTN model is more expressive than the basic STRIPS-like planning. On the other hand, the theoretical HTN model is not usable in practice, as it is undecidable, even under severe restrictions. HTN remains generally 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 of their execution[2].

As the theoretical model of HTN is undecidable, the computation could take an unlimited amount of time and we cannot even reliably predict the time in advance, so it is not (and cannot be) used in practice. In practice, modifications are used, restricting the plan space to finite and making the problem decidable [2].

The main HTN restrictions used are:

1. Restricting the length of a plan. As the maximal length of a plan becomes finite, the space of possible plans becomes finite, as we choose from a finite number of possibilities, when adding a task to the plan.
2. Restricting the methods to be acyclic. Any task can be expanded up to a finite depth, which is lower than the total number of tasks.
3. Restricting the task network to be totally ordered. Tasks are achieved serially, one after another, so subtasks cannot interleave.

Each of these restrictions alone is enough to make the HTN-like planning decidable, thus usable in practice. All current HTN-based planners use at least one of these restrictions (or their slight modifications). Therefore, it is better to use the term "HTN-like" planning for planning based on the HTN model with one of these three restrictions, rather than for the unrestricted theoretical model of HTN.

Naming convention. *The term "HTN-like planning" shall be used for planning based on the HTN model with a restriction, making its plan-space finite and the planning problem decidable.*

For the scope of this paper, we adhere to this naming convention.

Theorem 1. *Every HTN-like domain can be expressed as a STRIPS-like domain. Every STRIPS-like domain can be expressed as a HTN-like domain. Therefore, the expressivity of STRIPS-like planning is identical to the expressivity of HTN-like planning.*

Proof (sketch). The plan-space of HTN-like planning domain is finite; therefore the state-space of the domain is finite. For a finite state-space, we can construct a STRIPS-like domain by simply enumerating all possible state transitions as STRIPS actions. As a result, STRIPS-like planning expressivity is not smaller than the expressivity of HTN-like planning. The second half of the proof, showing that HTN-like planning expressivity is not smaller than the expressivity of STRIPS-like planning, is constructive, can be found in [2] and is based on the transformation of STRIPS-like domain to a flat HTN-like domain with decomposition depth 0. □

Enumerating all states of a domain is not very practical and leads to exponential number of actions. In the next sections, we will show how to transform a HTN-like domain into a STRIPS-like domain in low-order polynomial time, using STRIPS for emulating the HTN decomposition.

3 STRIPS as a Turing Machine with Finite Tape

The previous sections showed, that STRIPS-like and HTN-like planning expressivity is identical. In this section, we will provide a simple construction of a Turing machine with finite tape using STRIPS. This way, we show that STRIPS expressivity (and therefore also HTN-like planning expressivity) is equal to the expressivity of a Turing machine with finite tape.

Turing machine is a tuple:

$$M = (Q, \Gamma, b, \delta, q_0, F) \tag{1}$$

where Q is a finite set of states, Γ is a finite set of tape symbols, $b \in \Gamma$ is the blank symbol, $\delta = Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function (L and R are the shift left and right symbols respectively), q_0 is the initial state and $F \subseteq Q$ is the set of final states.

In the STRIPS emulation, the set of states Q will be represented by a set of constants C_Q. Γ is represented by set of constants C_Γ. Current state is expressed by the literal $state(q)$, position of the reading head by $position(p)$ and the symbol on a specific tape location is expressed as the literal $symbol(\gamma, p)$. The transition function is defined by the literal $transition(q, g, q', g', m)$, where $q \in C_Q$ and $g \in C_\Gamma$ are the old state and symbol on the tape, $q' \in C_Q$ and $g' \in C_\Gamma$ are the new state and tape symbol and $m \in \{LEFT, RIGHT\}$ is the direction of head movement. The following operators encode the Turing machine with finite tape:

```
operator: stateTransition
pre: state(q), position(p), symbol(g, p), transition(q, g, q', g', m),
```

```
   translate
del: state(q), symbol(g, p), translate
add: state(q'), symbol(g', p), move(m)

operator: moveHeadLeft
pre: move(LEFT), position(from), leftOf(to, from)
del: move(LEFT), position(from)
add: position(to), translate

operator: moveHeadRight
pre: move(RIGHT), position(from), leftOf(from, to)
del: move(RIGHT), position(from)
add: position(to), translate

operator: finish
pre: state(q), final(q), translate
del: state(q), translate
add: stop
```

As we can see, the machine operates in two steps: state transition and head movement. The machine stops if one of the final states is reached, marked with literal $final(q)$.

Prior to starting the STRIPS reasoning, we have to "create" the structure of the tape by adding literals $leftOf(left, right)$ for each two neighbouring states. With extensions of STRIPS allowing arithmetic operators, we would instead use the $+$ and $-$ operator to move right and left.

The tape sequence representation for a tape of length n uses $n-1$ literals. On the other hand, we have to remember, that the memory contents is also stored in the form of literals, so the n memory places are represented by $2n-1$ literals, increasing the memory complexity only constantly, comparing to the Turing machine.

The emulated finite tape Turing machine is deterministic if only one literal $transition(q, g, q', p', m)$ exists for some (q, g). Otherwise, the machine is non-deterministic.

We could also add input to the machine, represented in the same way like the tape. This would, of course, not change the expressivity.

Theorem 2. *The STRIPS domain defined above emulates a Turing machine with finite tape. The emulation time complexity is constantly higher than the complexity of the emulated machine.*

Proof (sketch). The initial literal set is $state(q_0)$ together with the encoding of the Turing machine as described above. The final condition for the STRIPS planner is set to *stop*. It is easy to see, that if the Turing machine stops, the STRIPS planner creates a plan leading from the initial to the final state and each two following steps of the plan (stateTransition, moveHeadLeft/Right) correspond to one step of the Turing machine. If the finite tape Turing machine doesn't stop, no plan is found. If the emulated machine is deterministic, exactly one action is

executable at each state and the plan derivation process is deterministic. If the emulated machine is non-deterministic, the STRIPS planner chooses one action for execution at each state, the same way the machine has to choose one. If the decision doesn't lead to the final state, the planner backtracks and systematically searches all alternatives until the final state is found or no more alternatives are left (i.e. the machine doesn't stop). □

Turing machine with an *in*finite tape is an important theoretical concept. On the other hand, the computers we use in practice have only finite memory and can be simulated by a Turing machine with finite tape.

The equality of expressivity of STRIPS and Turing machine with finite tape has an important implication. It means that STRIPS can express all problems solvable by a computer. The expression as a STRIPS domain can, of course, be sometimes very artificial and clumsy and computational complexity can be much larger. Nevertheless, STRIPS expressive power should not be underestimated.

4 STRIPS as HTN Emulator

The previous section shows, that it is possible to express a finite tape Turing machine as a STRIPS domain. Together with the possibility to express a HTN-like domain (with restrictions to be decidable) using the finite tape Turing machine, it is obvious, that STRIPS can express an arbitrary HTN-like domain.

This section provides a conversion of a HTN-like domain to STRIPS in low-order polynomial time. The conversion is based on emulating the decomposition of tasks by STRIPS plan derivation. The final STRIPS plan expresses the order of decomposition.

Let's say there is a task network $n = (A, \{B, C\})$, allowing the decomposition of task A into B and C. We create two operators A_{start}, A_{stop} and operators for B and C. A_{start} adds literals ($B_{A_{init}}$, $C_{A_{init}}$) allowing the execution of B and C. B can be decomposable again, so it again consists of operators B_{start}, B_{stop}, allowing its further decomposition. If B is a primitive task, it consists of only one operator B. After B is processed (operator B_{stop} or B finished), it adds literal $B_{A_{finish}}$, which is a part of A_{stop} precondition. This way, A_{stop} is only executed after all subtasks of A are fully decomposed or primitive. The initial state of the STRIPS planner contains only the literal S_{init}, allowing the start of the root task S (or possibly several literals if there are more root tasks). The final state contains the literal S_{finish}, which is reached after the full decomposition of the root task S and all its subtasks. Additionally, the final state may contain literals, added in tasks marked as goal tasks.

If task interleaving is allowed, then we have to avoid situations when a finished subtask allows the ending of a parent task from a different task network. Therefore, we have to create different operators for a subtask, which is in more task networks. For the same reason, we have to create separate operators for a parent task, which is in more task networks. As a result, a task being parent in n task networks and a subtask in m networks is converted to $m * n$ operators.

The HTN to STRIPS conversion algorithm for an acyclic decomposition graph is expressed by the following pseudocode. We use the shortened notation of an operator $Op = (pre, del, add)$, where *pre*, *del* and *add* are the precondition, delete and add sets of the operator *Op*. An overview of corresponding concepts of HTN-like and STRIPS-like domain after the conversion can be found in table 1.

```
for each task A
   if A is a decomposition of some task B, for each B
      if A is primitive
         add operator
            A_B = (A.pre ∪ {A_Binit}, A.del ∪ {A_Binit}, A.add ∪ {A_Bfinish})
      else
         for each task network n_i = (A, {C_j | j}) add operators
            A_iB_start = (A.pre ∪ {A_Binit}, {A_Binit}, {C_jA_init})
            A_iB_stop = ({C_jA_finish}, A.del ∪ {C_jA_finish}, A.add ∪ {A_Bfinish})
   else
      if A is primitive
         add operator
            A = (A.pre ∪ {A_init}, A.del ∪ {A_init}, A.add ∪ {A_finish})
      else
         for each task network n_i = (A, {C_j | j}) add operators
            A_i_start = (A.pre ∪ {A_init}, {A_init}, {C_jA_init})
            A_i_stop = ({C_jA_finish}, A.del ∪ {C_jA_finish}, A.add ∪ {A_finish})
for each initial task S
   add literal S_init to the initial state
   add literal S_finish to the goal state
```

Table 1. Corresponding concepts of HTN and STRIPS emulation of HTN

HTN	STRIPS
primitive task A	operator A
non-primitive task A	operators $A_{i_{start}}$, $A_{i_{stop}}$
task network $n_i = (A, \{C_j \mid j\})$	operators $A_{i_{start}}$, $A_{i_{stop}}$; $C_{jA_{i_{start}}}$, $C_{jA_{i_{stop}}}$ or C_{jA_i}
adding task A as a subtask of B	adding literal $A_{B_{init}}$ into the actual state
choosing one of the possible decompositions of A	choosing one applicable operator $A_{iB_{start}}$ with literal $A_{B_{init}}$ in the precondition
decomposing A using the task network $n_i = (A, \{C_j \mid j\})$	executing the sequence of operators $A_{iB_{start}}$; $C_{jA_{start}}$, $C_{jA_{stop}}$ or C_{jA}; $A_{iB_{stop}}$

If necessary, parameters (i.e. variables and constants) can be passed from the decomposed task A to its subtask C by adding parameters to the literal $C_{A_{init}}$. Other constraints among tasks, like task precedence or mutual exclusion can be simply expressed by adding literals to the operators. For example if we want operator A to precede operator B, we simply add a literal to the *add* part of A and to the *pre* part of B. This will prevent B being executed before A.

Theorem 3. *The algorithm above converts an acyclic HTN-like domain to an equivalent STRIPS-like domain. The STRIPS-like domain time complexity is constantly higher than the complexity of the initial HTN-like domain.*

Proof (sketch). The plan derivation in the resulting STRIPS-like domain copies the decomposition of the initial HTN-like domain. For every decomposition of some task A (i.e. choosing a suitable task network and adding its subtasks $\{B_j\}$ to the plan), there is exactly one A_{start} action, one action $B_{j\,A}$ for every primitive task from $\{B_j\}$, one literal $B_{j\,A_{start}}$ for every non-primitive task from $\{B_j\}$ and one A_{stop} action. Actions representing subtasks of A are never executed before A_{start} or after A_{stop}. The STRIPS planning algorithm only chooses actions at a point, when a HTN-like planner would choose a task to decompose and a task network to be used for the decomposition. This means, we have exactly one STRIPS action for every step of a HTN-like planner and exactly one decision of the STRIPS planner for every decision of the HTN-like planner. □

If we want to use cyclic decomposition graphs (i.e. a task can be decomposed into itself after some steps), we have to restrict the domain to a maximal depth of decomposition or to be fully ordered (see section 2 Expressivity) in order to have a decidable domain.

For a fully ordered domain, we simply add precedence restrictions on operators.

If we want to restrict the maximal depth of decomposition while having a cyclic and not fully ordered domain, we have to mark the actions to differentiate decompositions of the same task by the same task network in a cyclic decomposition. This can be done by creating a sequence of symbols (like the finite tape in the previous section), which are then used for marking actions representing one decomposition. We simply add an incrementing action (similar to the moveHeadRight from the previous section) after each A_{start} action, while the current "counter" value is a parameter of A_{start} and this value is carried between operators representing the same task network using the $B_{A_{init}}$ and $B_{A_{finish}}$ literals. This finite set of marking symbols is the equivalent of task marking symbols used in HTN-like planners.

Many planners (based on HTN or STRIPS) allow different extensions to the basic HTN, like handling resources, allowing variables and arithmetic operators, allowing concurrency, timed actions or planning with uncertainty. According to Theorem 2, all extensions of HTN-like planning (as long as the problem remains computable) can be transformed to STRIPS, perhaps increasing computational complexity. On the other hand, most extensions are common for both approaches, so it is possible to modify the conversion algorithm introduced in this section to achieve the same time complexity of the planning process.

5 Conclusions

The concept of HTN is nothing more (but nothing less) than allowing the user to provide the planning engine with additional heuristic information about how to construct a plan, but does not increase the domain-space.

Nevertheless, HTN is a very useful and user-friendly concept, as we can see on the large number of practical uses.

This paper shows that STRIPS-like planning can be used for the same domains, HTN-like planning (with restrictions causing it to be decidable) can be used for (i.e. the expressivity of both approaches is identical). Moreover, the domains can be converted from HTN-like to STRIPS-like and vice versa in low order polynomial time, thus allowing the theoretical results for STRIPS-like planning to be used for HTN-like planning and vice versa.

Finally, this paper shows that STRIPS expressivity is equal to the expressivity of a Turing machine with finite tape, i.e. all problems that can be solved by a (common) computer can also be solved by STRIPS. This is rather a theoretical result than a practically usable conversion. However, it places the lower and upper bound on both, STRIPS-like and HTN-like planning expressivity. Additionally, complexity results for different STRIPS-like and HTN-like domains can make use of Turing machines formalism.

Acknowledgments. This work was partially supported by the Slovak State Programme of Research and Development "Establishing of Information Society" under contract No. 1025/04; by the Science and Technology Assistance Agency under the contract No. APVT 51-024604; by the Grant Agency of Slovak Republic grant No. VG 1/3102/06.

References

1. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence 69, Elsevier Science Publishers Ltd., Essex, UK (1994) 161–204.
2. Erol, K., Nau, D., Hendler, J.: HTN planning: Complexity and expressivity. Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94) AAAI Press, Menlo Park, USA (1994)
3. Fikes, R.E., Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, Elsevier Science Publishers Ltd., Essex, UK (1971) 189–208.
4. Kambhampati, S., Mali, A., Srivastava, B.: Hybrid planning in partially hierarchical domains. Proceedings of the National Conference on Artificial Intelligence (AAAI) (1998)
5. Lesser, V.R., et al.: Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. Autonomous Agents and Multi-Agent Systems. Vol. 9, No. 1, Kluwer Academic Publishers (2004) 87–143.
6. Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research 20 (2003) 379–404
7. Sacerdoti, E.D.: A Structure for Plans and Behavior. Elsevier-North Holland (1977)
8. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research. Vol. 7 (1997) 83–124.
9. Zalaket, J., Camilleri, G.: FHP: Functional Heuristic Planning. 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Wellington, New Zealand, Springer-Verlag (2004)

N.T. Nguyen et al. (Eds.): KES-AMSTA 2007, LNAI 4496, pp. 121-130, 2007.

http://www.springerlink.com/content/w240254584rn317n/