Translating Totally Ordered HTN Planning Problems to Classical Planning Problems Using Regular Approximation of Context-Free Languages

Daniel Höller
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
hoeller@cs.uni-saarland.de

Abstract

There have been several approaches to use techniques from classical planning in HTN planning. While a direct translation is in general not possible due to the different expressiveness, there have been translations of *bounded* HTN problems and approaches to use classical heuristics in HTN search procedures. In this paper, we introduce a different approach. We exploit methods from the field of Computational Linguistics introduced to approximate Context-Free Languages by Finite Automata. We use them to approximate the decomposition structure of Totally Ordered (TO) HTN planning problems by classical problems. The resulting problem can then be solved using standard classical planning systems. A subset of TO-HTN problems can be translated exactly, i.e., without changing the set of solutions. For problems where an approximation is necessary, we use an *over*approximation, i.e., the set of solutions to the classical problem is a superset of that of the HTN problem. We then use plan verification to check whether a solution is valid and thus obtain a sound and complete overall approach. The resulting system outperforms the state of the art on the IPC 2020 benchmark set in terms of coverage.

1 Introduction

Classical planning problems consist of a propositional environment model and a set of actions specifying how to change it. Actions come with preconditions that need to be satisfied to apply them, and effects describing the change caused by the application – implicitly defining a state transition system. Planners need to find an action sequence transforming a given initial state into one where certain goal properties hold. Hierarchical Task Network (HTN) (Erol, Hendler, and Nau 1996; Bercher, Alford, and Höller 2019) planning additionally specifies a grammar-like decomposition structure: *abstract* tasks cannot be applied directly, but need to be decomposed into other tasks using *decomposition methods* until only actions are left. To solve the problem, an initial task needs to be decomposed in an applicable action sequence.

The motivations to use HTN planning are manifold: While classical planning problems can express structures equal to regular languages, HTN planning can express up to context-sensitive languages (Höller et al. 2014, 2016). Further, some problems can be modeled in a more intuitive way using the hierarchy (Goldman 2009), and the hierarchy has also been used to enable communication on different levels of abstraction (Behnke et al. 2020a; Köhn et al. 2020), or to add search control to the model (Nau et al. 2003).

For models without (sufficient) search control, planning systems need to use sophisticated search techniques to find a solution. While such techniques are well-studied in classical planning, there is not much work in HTN planning, making it appealing to use classical techniques in HTN planning. The family of *Relaxed Composition* heuristics can apply classical heuristics directly to guide the search in HTN planning (Höller et al. 2018b, 2019, 2020b). Other techniques have been extended, e.g. propositional encodings for SAT-based planners (Behnke, Höller, and Biundo 2018, 2019; Schreiber et al. 2019), or IP encodings for heuristics (Höller, Bercher, and Behnke 2020). Alford et al. (2009; 2016) introduced translations from HTN problems into classical problems by adding a new part to the state of the problem representing the decomposition process. Due to the higher expressiveness, this is only possible for *bounded* HTN problems.

In this paper, we present a novel translation of HTN planning problems to classical problems for *Totally Ordered* (TO) HTN planning, where tasks in decomposition methods are totally ordered. In this subclass, methods are equivalent to rules of a context-free grammar (CFG). Our approach is based on techniques to approximate CFGs by Finite Automata (FA) (Nederhof 2000a,b). We use them to construct a FA that accepts those sequences of actions that may result from the decomposition process. We then create a classical problem that includes the state translation system of the original HTN problem and the FA. This problem can be solved using standard classical planners. For a subclass of TOHTN problems, this can be done without changing the set of solutions. For the rest we use an *over*approximation of the solution set and verify solutions found by the classical planner. Both variants result in a sound and complete overall approach, though our current implementation[1] is *in*complete due to technical reasons. Our evaluation on the 2020 International Planning Competition (IPC) benchmarks shows that the majority of instances can be translated exactly. Our system outperforms the IPC participants and several systems from the literature in terms of coverage.

[1]Source code is available online *toad.hierarchical-task.net*

2 Formal Framework

We use the formalism by Geier and Bercher (2011) in a variant tailored to TOHTN (Behnke, Höller, and Biundo 2018).

A classical problem is a tuple $p = (L, A, s_0, g, \delta)$. L is a finite set of propositional state features. A state s is defined as the set of those state features that hold in it, i.e. $s \in 2^L$. s_0 is the initial state of the problem, $g \in 2^L$ is the goal condition. A state s is a goal state if and only if $g \subseteq s$. A is a finite set of actions. They are mapped to their preconditions, add effects and delete effects by the functions given in $\delta = (prec, add, del)$, $\{prec, add, del\} : A \to 2^L$. An action a is applicable in a state s if and only if $prec(a) \subseteq s$. The state s' resulting from the application of a in s is defined as $s' = \gamma(a, s) = (s \setminus del(a)) \cup add(a)$. A sequence of actions $a_1, a_2, \ldots, a_n$ is applicable in a state s_1 if and only if a_i is applicable in s_i with $s_i = \gamma(a_{i-1}, s_{i-1})$ for $i \leq 2 \leq n+1$. We denote s_{n+1} the state the application *results* in. A solution to a problem is a sequence of actions that is applicable in s_0 and results in a goal state.

An HTN problem is a tuple $p = (L, C, A, M, s_0, c_I, g, \delta)$, where L, A, s_0, g and δ are defined as before. In the context of HTN planning, actions are also called *primitive tasks*. We use these terms synonymously. C is a finite set of abstract (or compound) tasks with $C \cap A = \emptyset$. Tasks are maintained in *task networks*. In *Totally Ordered* HTN planning, a task network is just a sequence of tasks. Let $\mathcal{T} = C \cup A$. A task network is an element out of $\mathcal{T}^*$ (where * is the Kleene operator). Abstract tasks are decomposed using *(decomposition) methods*. A method maps an abstract task to a task network. The set M contains all methods, i.e. $M \subseteq C \times \mathcal{T}^*$. A method (c, φ) is applicable to the abstract task c. When it is applied to a task network $\omega\, c\, \omega'$ with $\varphi, \omega, \omega' \in \mathcal{T}^*$ and $c \in C$, the resulting task network is defined as $\omega\, \varphi\, \omega'$. We write $tn \to^* tn'$ to denote that tn can be decomposed into tn' by 0 or more method applications.

c_I is the initial task. A solution to a TOHTN planning problem is a task network ω with

1. $c_I \to^* \omega$, i.e., it can be reached by decomposing c_I,
2. $\omega \in A^*$, i.e., all tasks are primitive,
3. ω is applicable in s_0. It results in a goal state[2].

3 Translating TOHTN to Classical Problems

The definition of HTN solutions maps the problem to a (maybe infinite) set of action sequences, the solutions. This is just like a formal grammar defines a formal language (Höller et al. 2014). The set of actions forms the set of terminal symbols over which the language is defined. As has already been observed before (Höller et al. 2014), the language formed by the solutions to the overall problem can be seen as the intersection of two other languages, one defined by the decomposition hierarchy, i.e., by the abstract tasks and methods, and one defined by the state transition system defined by the preconditions and effects of the actions:

$$L_H = \{\omega \in A^* \mid c_I \to^* \omega\}$$
$$L_C = \{\omega \in A^* \mid \omega \text{ appl. in } s_0 \text{ and results in a goal state.}\}$$

[2]In HTN planning, the goal definition is usually empty. Whether or not there is one does not matter for our approach.

The first language imposes constraints introduced by solution criterion 1 and 2, the second one those imposed by solution criterion 3. Action sequences that are included in both languages are in the set of solutions[3]. The basic idea of our approach is to construct a FA that accepts a superset of L_H and encode it on top of the actions that already define L_C.

An overview is given in Figure 1. We first only consider the language L_H, i.e., we ignore the state and preconditions and effects of actions and consider the decomposition structure to be a CFG (the respective edge is marked with *(a)* in the figure). Though the decomposition rules syntactically resemble exactly a CFG, this does not mean that the encoded language is actually context-free, it might be regular. We check a sufficient condition that holds if the grammar describes a regular language *(b)*, which is described in detail in Section 3.1. If it does not, we change the model using a method introduced by Nederhof (2000b) in a way such that (1) it becomes a regular language and (2) the described language is a superset of L_H *(c)* (Section 3.3). Whether or not it is an approximation – we now have a CFG describing a regular language, which we encode into a FA *(d)* (Section 3.2) using a method introduced by Nederhof (2000a). The labels of the FA are actions from the HTN domain.

So far, the FA encodes L_H (with or without approximation). Now we need to intersect it with L_C. We do this implicitly by encoding a classical problem that holds one action for each transition of the FA *(f)* (Section 3.4). Each introduced action has preconditions and effects encoding the state change in the FA as well as its original ones. The resulting classical problem is solved using a classical planner. When we did no approximation, any plan returned by the planner is also a solution to the HTN problem. When we *did* an approximation, the set of solutions to the classical problem is a superset of the one of the HTN problem and we cannot directly return a solution. Instead, we first verify plans found by the classical planner to ensure that they are also plans for the HTN problem. Since we *over*approximate the set of solutions, our approach is complete (though our implementation is not, we will discuss this in Section 5).

Let $p = (L, C, A, M, s_0, c_I, g, \delta)$ be an HTN planning problem. The language L_H is defined by the elements $G_H = (C, A, M, c_I)$ that syntactically form a CFG with the non-terminal symbols C, the terminal symbols A, the production rules M, and the start symbol c_I. Its semantics – defined by the HTN solution criteria 1 and 2 – are also equivalent to the derivation rules of formal grammars. We apply the following transformation to a FA accepting L_H on this grammar.

3.1 Problem Analysis

We now describe a criterion introduced by Nederhof (2000a) that we use to decide if we need to approximate the HTN problem or not. He gives the following definition (p. 19):

Definition 1 (Self-embedding grammars). *A CFG $G_H = (C, A, M, c_I)$ is self-embedding when there is some $c_a \in \mathcal{T}$ such that $c_a \to^* \alpha c_a \beta$ with $\alpha \neq \varepsilon$ and $\beta \neq \varepsilon$.*

[3]The languages are defined with respect to a given HTN problem p. However, in the following it will be clear from the context to which problem we refer to and we will just write L_H and L_C.

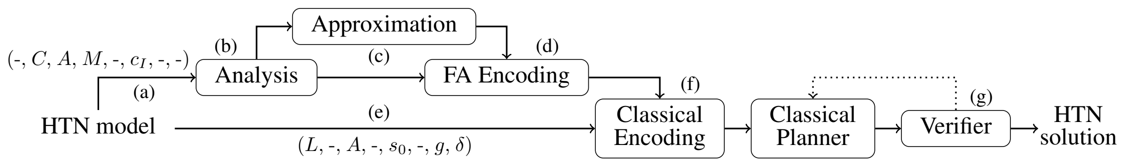

Figure 1: Overview of the approach.

When a CFG is *not* self-embedding, it describes a regular language (Nederhof 2000a, p. 19) and we can directly apply the translation to the FA. Intuitively, this is because a recursive cycle in the decomposition structure then only adds symbols to the left, *or* to the right.

Nederhof (2000a, p. 19) introduces the following test on whether a CFG is self-embedding. Let $G_H = (C, A, M, c_I)$ be a CFG, $\bar{N} = \{c_a \in C \mid \exists \alpha, \beta : c_a \rightarrow^* \alpha c_a \beta\}$ the set of recursive symbols, and $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ a partition of the recursive symbols such that $c_a, c_b \in \bar{N}$ are in the same partition if and only if $\exists \alpha_1, \beta_1, \alpha_2, \beta_2 : c_a \rightarrow^* \alpha_1 c_b \beta_1 \wedge c_b \rightarrow^* \alpha_2 c_a \beta_2$. I.e., we calculate the strongly connected components of the grammar.

Definition 2. *Some $N_i \in \mathcal{N}$ is left generating (we will write $lg(N_i)$) if and only if $\exists (c_a, \alpha c_b \beta) \in M$ such that $c_a, c_b \in N_i$ and $\alpha \neq \varepsilon$, it is right generating ($rg(N_i)$) if and only if $\exists (c_a, \alpha c_b \beta) \in M$, $c_a, c_b \in N_i$ and $\beta \neq \varepsilon$.*

Definition 3 (Left/right recursive, cyclic)**.** *Some $N_i \in \mathcal{N}$ is left recursive if and only if $\neg lg(N_i)$ and $rg(N_i)$. It is right recursive if and only if $lg(N_i)$ and $\neg rg(N_i)$. It is cyclic if and only if $\neg lg(N_i)$ and $\neg rg(N_i)$.*

Definition 4 (Self recursive)**.** *Some $N_i \in \mathcal{N}$ is self recursive if and only if $lg(N_i)$ and $rg(N_i)$.*

When there is some N_i that is self recursive, this is a necessary and sufficient condition for G_H to be self-embedding (Nederhof 2000a, p. 20). For self-embedding problems, the test identifies which rules need to be modified, namely those causing G_H to be self-embedding.

We first introduce the exact translation of non-self-embedding models and the approximation afterwards.

```
1  procedure make_fa($q_0$, $\alpha$, $q_1$)
2      if $\alpha = \varepsilon$ then $\Delta = \Delta \cup \{(q_0, \varepsilon, q_1)\}$
3      else if $\alpha = a$, $a \in A$ then $\Delta = \Delta \cup \{(q_0, a, q_1)\}$
4      else if $\alpha = x\beta$, $x \in \mathcal{T}$, $\beta \in \mathcal{T}^*$, $|\beta| > 0$ then
5          $q$ = fresh_state
6          make_fa($q_0$, $x$, $q$)
7          make_fa($q$, $\beta$, $q_1$)
8      else
9          $c_a = \alpha$    /* $\alpha$ is abstr. task */
10         if $\exists i : c_a \in N_i$ then
11             for $c_b \in N_i$ do $q_{c_b}$ = fresh_state
12             if recursive$(N_i)$ = left then
13                 for $(c_c, x_1 \ldots x_m) \in M$ s.t. $c_c \in N_i$
                    $\wedge\, x_1, \ldots, x_m \notin N_i$ do
14                     make_fa($q_0$, $x_1 \ldots x_m$, $q_{c_c}$)
15                 for $(c_c, c_d x_1 \ldots x_m) \in M$ s.t.
                    $c_c, c_d \in N_i \wedge x_1, \ldots, x_m \notin N_i$ do
16                     make_fa($q_{c_d}$, $x_1 \ldots x_m$, $q_{c_c}$)
17                 $\Delta = \Delta \cup \{(q_{c_a}, \varepsilon, q_1)\}$
18             else
19                 for $(c_c, x_1 \ldots x_m) \in M$ s.t.
                    $c_c \in N_i \wedge x_1, \ldots, x_m \notin N_i$ do
20                     make_fa($q_{c_c}$, $x_1 \ldots x_m$, $q_1$)
21                 for $(c_c, x_1 \ldots x_m c_d) \in M$ s.t.
                    $c_c, c_d \in N_i \wedge x_1, \ldots, x_m \notin N_i$ do
22                     make_fa($q_{c_c}$, $x_1 \ldots x_m$, $q_{c_d}$)
23                 $\Delta = \Delta \cup \{(q_0, \varepsilon, q_{c_a})\}$
24         else
25             for $(c_a, \beta) \in M$ do
26                 make_fa($q_0$, $\beta$, $q_1$)
```

Figure 2: Algorithm by Nederhof (2000a, Figure 2) to transform non-self-embedding CFGs to a FA.

3.2 Non-Self-Embedding Problems

Let $F = (Q, \Sigma, \Delta, q_I, q_F)$ be a FA with the set of states Q, the set of terminal symbols Σ, the set of transition rules $\Delta \subseteq Q \times \Sigma \times Q$, the initial state q_I, and the set of final states q_F. We translate $G_H = (C, A, M, c_I)$ of non-self-embedding HTN problems to a FA using the algorithm of Nederhof (2000a, Figure 2). It is given in Figure 2, with only minor adaptations in the notation. It is initially called with the initial task c_I. From an abstract point of view, it goes down the hierarchy, collecting all possible sequences of actions that may result from decomposition.

It gets three arguments: two states of the FA q_0 and q_1, and a sequence of tasks $\alpha \in \mathcal{T}^*$. Intuitively, α is a sequence of tasks that may come between the two states.

When α is empty, an ε-transition from q_0 to q_1 is added to the FA (line 2). When it is a single action, a new transition q_0 to q_1 labeled with α is added (line 3). When α is a sequence of at least 2 tasks, a new state q is created as intermediate state and the algorithm is recursively called: once with q_0 and q and α's first symbol and once with q and q_1 and the rest of α (line 4-7). When none of these cases applies, α consists of a single abstract task c_a. Such a task may be recursive and thus be part of some N_i (line 10-23) or not (line 24-26). In the latter case, the next tasks to come are those from the methods to decompose c_a. For each such method the algorithm is called. When c_a is part of some N_i, there are two cases: the N_i might be left recursive or right recursive. First, a new state is created for each task that is part of N_i (line 11). In the left recursive case, the recursion is through the left most task, the tasks added last are executed first. The rules leaving the recursion are added in line 13-14. Recursive methods result in transitions between the newly introduced states (line 15-16). From the task c_a that formed

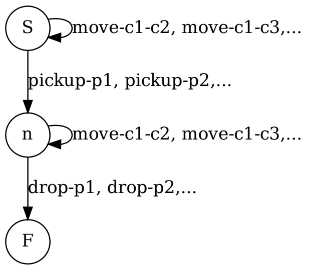

Figure 3: Transport FA.

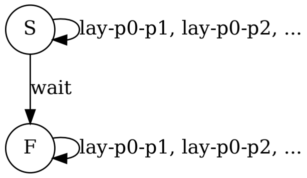

Figure 4: Roadie Robot FA.

our α, an ε-transition goes to the state q_1. The right recursive case is analog, now starting with the recursive methods (line 21-22), and ending with the ones leaving N_i (line 19-20).

Example 1. *Consider a transport domain with the following set of methods. Tasks starting with a capital letter are abstract, tasks starting with a non-capital letter are primitive. The initial task is D (deliver a package).*

$$\begin{aligned}M = &\{(D, (MovetoP, L, MovetoD, U))\}\\ &\cup \{(MovetoP, (move\text{-}c_1\text{-}c_2, MovetoP)) \mid c_1, c_2 \in C\}\\ &\cup \{(MovetoD, (MovetoD, move\text{-}c_1\text{-}c_2)) \mid c_1, c_2 \in C\}\\ &\cup \{(MovetoP, \emptyset), (MovetoD, \emptyset)\}\\ &\cup \{(L, (pickup\text{-}p)), (U, (drop\text{-}p)) \mid p \in P\}\end{aligned}$$

D is decomposed (line 1) into tasks to move to the package, load it, move to the destination, and to unload it there. The two abstract move tasks are recursive (line 2 and 3). We made one rule left, and one right generating (just to show that this has no effect on the FA). One recursion of an abstract move task leads to a single move action (e.g. $move\text{-}a\text{-}b$), followed (or preceded) by the abstract task. The resulting FA is shown in Figure 3 (ε-transitions have been removed). L_H contains words that start with a (possibly empty) series of move actions. Be aware that this does not mean that there is a single move action that is actually applicable *in the road network. Then, a package is picked up, followed by another series of move actions and a drop.*

Unfortunately, Nederhof (2000a) does not give an analysis of the FA's size, so we do this in the following. We are interested in the number of transitions, because they lead to actions in our final model. We start with non-recursive problems and discuss recursion afterwards. We construct a graph that describes the paths that the recursive calls of make_fa follow. Its root is the initial task c_I. For every method (c, ϕ) applicable to c_I, we add a new node labeled x and an arc (c, x) for every $x \in \phi$. Now we do the same for all x, and further continue the process until all task nodes without outgoing edges are primitive.

For each method, make_fa adds states and transition rules. When a method has n subtasks, it builds n subautomata and connects them (via lines 6-7), so the method itself adds n transitions to the size of the subautomata. The leafs of the graph are actions that add only a single transition. Let ms be the number of subtasks for the method with the most subtasks (max size). Let h be the height of the tree, and mm the maximum number of methods for a

1. Add new non-terminals $a_b^{\uparrow}$ $a_b^{\downarrow}$ $a_b^{\leftarrow}$ $a_b^{\rightarrow}$ with $a, b \in N_i$
2. Add the following methods with $a, b, c, d, e \in N_i$

$$\begin{aligned}&(a, a_b^{\uparrow}) && (1)\\ &(a_b^{\uparrow}, a_c^{\leftarrow} x_1 \ldots x_m c_b^{\downarrow}), \forall (c, x_1 \ldots x_m) \in M && (2)\\ &(a_b^{\downarrow}, c_a^{\rightarrow} x_1 \ldots x_m e_b^{\uparrow}), \forall (d, \alpha c x_1 \ldots x_m e \beta) \in M && (3)\\ &(a_b^{\downarrow}, b_a^{\rightarrow}) && (4)\\ &(a_b^{\leftarrow}, x_1 \ldots x_m c_b^{\leftarrow}), \forall (a, x_1 \ldots x_m c \beta) \in M && (5)\\ &(a_a^{\leftarrow}, \varepsilon) && (6)\\ &(a_b^{\rightarrow}, c_b^{\rightarrow} x_1 \ldots x_m), \forall (a, \alpha c x_1 \ldots x_m) \in M && (7)\\ &(a_a^{\rightarrow}, \varepsilon) && (8)\end{aligned}$$

3. Remove (a, α) from M

Figure 5: Transformation of self-embedding N_is proposed by Nederhof (2000b, p. 9). a-e denote tasks part of N_i, all x_is are not part of N_i, α and β are sequences of tasks (that might or might not be part of N_i).

task (max methods). The size of the FA is then bounded by $(ms \times mm)^h$. Now consider a recursive grammar. For a task inside some N_i, the algorithm adds one state for each task in N_i, and processes each method that decomposes a task in N_i. Methods leaving N_i (lines 13-19) cause the transitions for n subautomata for a task further down in the graph to be added, where n is the size of the method. Methods leading to recursion cause such $n - 1$ subautomata to be added between two new states representing N_i. Be aware that (by construction of $\bar{N}$) the outgoing edges cannot lead to paths that, again, reach the same N_i. One additional transition is added when entering N_i. Let mN_i be the maximum number of methods decomposing tasks from the same N_i. We end up with a size bound of $(ms^2 \times mm \times mN_i)^h$ and it holds:

Proposition 1. *The size of the FA might be exponential in the size of the grammar.*

We see later in Sec. 6 that it is much smaller in practice.

3.3 Self-Embedding Problems

When G_H is self-embedding, we compile it using the over-approximation proposed by Nederhof (2000b, p. 9). The result is a new grammar G'_H that is not self-embedding with $L(G_H) \subseteq L(G'_H)$, where $L(G)$ is the language defined by the grammar G. For each N_i that is self recursive, the algorithm given in Figure 5 is applied. It disconnects the generation of the problematic parts of the rules into independent processes. Be aware that the αs and βs in line 3, 5, and 7 are not contained in the new rules. Therefore the parts between the symbols out of N_i can be generated separately.

Example 2. *Consider the Roadie Robot domain. A robot has to lay cables on a stage for a concert. Then, it waits until the concert is over, and re-collects the cables in the exact reverse order to avoid knots. The resulting solution set only includes palindromes (and thus is context-free).*

The method set of such a model could be as follows: $M = \{(LC, (lay\text{-}p_i\text{-}p_j, LC, lay\text{-}p_j\text{-}p_i)) \mid p_i, p_j \in waypoints\} \cup \{(LC, (wait))\}$. *In the FA resulting from approximation and* *translation (Fig. 4), the lay actions can be applied before and after the wait action without affecting each other.*

Again, we give an analysis of the resulting size. The variables a-e are universally quantified over the tasks in the N_i. However, some of the symbols are bound by the method set we are changing. The overall increase in grammar size is bounded by rule 3. Consider the pattern for the changed method, $(d, \alpha c x_1 \dots x_m e \beta)$. The right-hand side of it might match several times for one method. When there are k symbols from N_i, it matches $k-1$ times. Further, the variables a and b are universally quantified over the symbols in N_i. Let $|\mathcal{N}|$ be the number of partitions, mn the maximum size of the N_i sets, and mm the maximum number of methods for a task in N_i. Then, the blowup is bounded by $|\mathcal{N}| \times mn^2 \times mm \times (k-1)$ and the following holds:

Proposition 2. *The size of the new grammar is polynomial in the size of the original grammar.*

3.4 Translating FA to Classical Problems

We now have a FA defining sequences of actions that might be the result of the decomposition process. Next we define a classical problem such that its set of solutions contains a plan if and only if (1) it is accepted by the automaton, (2) it is applicable using the preconditions and effects defined in the original HTN problem p, and (3) the application results in a goal state from p.

Let $p = (L, C, A, M, s_0, c_I, g, \delta)$ with $\delta = (prec, add, del)$ be the original HTN planning problem and $F = (Q, \Sigma, \Delta, q_I, q_F)$ be the automaton constructed before.

We define the classical planning problem $p' = (L', A', s'_0, g', \delta')$ with $\delta' = (prec', add', del')$ as follows[4]:

$$L' = L \cup Q, s'_0 = s_0 \cup \{q_I\},\ g' = g \cup q_F$$
$$A' = \{a^q_{q'} \mid (q, a, q') \in \Delta\}$$

For the new actions $a^q_{q'} \in A'$, we define preconditions and effects as $prec'(a) = prec(a) \cup \{q\}$, $add'(a) = add(a) \cup \{q'\}$ and $del'(a) = del(a) \cup \{q\}$.

Theorem 1. *Every sequence $a1^{q1}_{q1'}, a2^{q2}_{q2'}, \dots, an^{qn}_{qn'}$ of actions that is a solution to p' is accepted by F.*

Proof. The set of states features Q from F forms a subset of L'. In s'_0, exactly one of these state features holds: the initial state of the automaton. Every action has the form $a^q_{q'}$, due to the newly introduced preconditions and effects, we know that in each (reachable) state of the problem, exactly one of the state features out of Q holds. An action $a^q_{q'}$ is contained if and only if $(q, a, q') \in \Delta$. Since $a1^{q1}_{q1'}$ is the first action we know that $q1 = q_I$ and that $(q1, a1, q1') \in \Delta$. Since $a2^{q2}_{q2'}$ is applicable afterwards, we know that $q1' = q2$ and that $(q2, a2, q2') \in \Delta$ and so on. Finally, since q_F is part of the state-based goal, we know that $qn' = q_F$ and thus that the automaton accepts the sequence[5]. □

[4]Wlog, we assume that $L \cap Q = \emptyset$.

[5]Note that the automaton has by construction a single final state.

Theorem 2. *For any sequence $\pi' = a1^{q1}_{q1'}, a2^{q2}_{q2'}, \dots, an^{qn}_{qn'}$ that is a solution to p', the sequence $\pi = a1, a2, \dots, a_n$ in the problem p is applicable and reaches a goal state.*

Proof. By construction $L' = Q \cup L$ and $Q \cap L = \emptyset$, i.e., the state features can be separated into two disjunctive sets such that one is equal to the state features of the original problem.

For the initial states of p and p', it holds that $s'_0 \setminus Q = s_0$. For each action, it holds that $prec'(ai^{qi}_{qi'}) \setminus Q = prec(ai)$. Therefore, when $a1^{q1}_{q1'}$ is applicable in s'_0, $a1$ is in s_0. All (add- and delete-) effects added to the original ones in the definition are from Q, i.e., $add'(ai^{qi}_{qi'}) \setminus Q = add(ai)$, and $del'(ai^{qi}_{qi'}) \setminus Q = del(ai)$. Therefore, for $1 \leq i \leq n$, it holds that when s'_i is the intermediate state after executing action i in π', $s_i \setminus Q$ is the one in π and thus $prec(a_{i+1}) \subseteq s_i$. The same holds for the goal condition. □

3.5 Properties of the Overall Approach

When using a sound classical planner that eventually returns all possible solutions for the generated problem, the following properties hold for the overall approach.

Theorem 3 (Soundness)**.** *The approach is sound.*

Proof. When we apply plan verification, we trivially know that a returned plan is valid. However, for problems that are not self-embedding, the approach is also sound without verification. Let FA be an automaton resulting from the translation of a non-self-embedding problem. From Theorem 1 we know that the solution we return is accepted by the FA, i.e., it fulfills solution criteria 1 and 2 from the HTN definition. From Theorem 2, we know that it fulfills criteria 3. I.e., it fulfills all solution criteria of the HTN definition. □

Theorem 4 (Completeness)**.** *The approach is complete.*

Proof. Let p be an HTN planning problem and p' a classical problem constructed as given above. Let $\pi = (a1, a2, \dots, an)$ be an arbitrary fixed solution to p. We need to show that π is a solution to p'.

We know that π is applicable and leads to a goal state in p. Since the new actions in p' are copies of those in p with respect to all state features apart from Q, we know that it is applicable and leads to a goal in p' when ignoring Q. What we need to show is that it is also applicable and goal-leading with respect to the preconditions and actions from Q.

Let L'_H be the language accepted by the constructed FA, whether or not it is an approximation, we know that $L'_H \supseteq L_H$ (Nederhof 2000a,b). I.e., π is accepted by the automaton and there is a sequence of FA states $(q_I, q_1, q_2 \dots, q_{n-1}, q_F)$ such that $\{(q_I, a1, q_1), (q_1, a2, q_2), \dots, (q_{n-1}, an, q_F)\} \in \Delta$. Then by construction, the action sequences $\pi' = (a1^{q_I}_{q_1}, a2^{q_1}_{q_2}, \dots, an^{q_{n-1}}_{q_F})$ is applicable in p', which results in the finite state of the FA. □

4 Discussion and Related Work

We have been asked by the reviewers to discuss whether it is interesting to compare our work with approaches to integrate control knowledge into classical models (see e.g. Baier, Fritz, and McIlraith (2007) or Chrpa and Barták (2016)).

Though there are systems (like SHOP2, Nau et al. (2003)) that exploit HTN models just in this way with great success, this is not our view on HTN planning. As given in the introduction, the motivations to use HTN planning are manifold and adding advice is just one of them. What insights would we gain when we beat a classical planner or other approaches to add control knowledge with our system? Whether or not we are able to encode helpful advice. HTN planning adds a second means of introducing constraints on the set of solutions to a planning problem. Like the state in classical planning, this can be used to encode physics *or* advice. This view is also reflected in the benchmark sets of the 2020 IPC: they are diverse and include e.g. domains learned from classical domains, domains created as HTN models without advice in mind (e.g. Assembly or Entertainment), and even undecidable problems (PCP). We want our system to be domain-independent: it shall find a solution for a given model – with or without advice. Therefore this is the focus of both the discussion of related work and the evaluation.

Alford et al. (2009; 2016) introduced a translation from HTN problems to classical problems. They also consider partially ordered HTN planning, but we only discuss their closer related TO encoding. They simulate a HTN progression search in the state of the classical problem. In progression search, only tasks without predecessors in the current task network are processed. In the TO setting, there is always a single first (primitive or abstract) task. When it is primitive, it is either applicable or not, and thus can be applied and removed, or the entire task network cannot be refined anymore. When it is abstract, it is replaced by other tasks. Alford et al. model the tasks to process as a stack that is stored the classical problem's state. The top-most task is the first task of the task network. When it is decomposed, the new tasks are put on top and the stack size increases. When an action is applied, it decreases. The goal is to reach an empty stack. The approach is similar to parsing using a pushdown automaton, so it is quite different from ours. Other than in our encoding, the hierarchical structures like abstract tasks and methods are still present in the classical problem. In our approach, they are blended into the action set.

A second major difference is how the approaches overcome the smaller expressiveness of classical planning: Alford et al. bound the maximum size of task networks during search, while we use an overapproximation in combination with plan verification, so that we do not need to plan more than once (as necessary in a bound-based approach). There are some interesting connections between the property of *tail-recursiveness* introduced by Alford et al. to determine upper bounds for a certain subclass of problems and *(non) self-embedding* used here. However, a detailed discussion is out of scope for this paper.

5 Implementation

We use the 2020 IPC's input language HDDL (Höller et al. 2020a) and ground the models using the PANDA grounder (Behnke et al. 2020b). We then apply our translation. The classical problem is solved using Fast Downward (FD) (Helmert 2006). We found that simple configurations worked well and included two of them in the evaluation: one using greedy best first search (GBFS) in combination with the FF heuristic (Hoffmann and Nebel 2001) and preferred operators, and one that does Enforced Hill Climbing (EHC) with FF and preferred operators and (in case of failure) GBFS afterwards. We verify our plans using the compilation by Höller et al. (2018a). The verification problem is thereby compiled into a new HTN problem solved by an HTN planner, we used PANDA (Höller et al. 2018b). We verified all generated solutions and included grounding and verification in the overall runtime. Though this is not necessarily for the exact translation, verification rebuilds the decomposition steps applied to reach the solution, these might be interesting for some users, e.g. for plan explanation (Behnke et al. 2020a). We named our system TOAD (<u>T</u>otally <u>O</u>rdered HTN <u>A</u>pproximation using <u>D</u>FA).

domain	#inst	¬rec	¬self embedding			s.e.	ukn
			l rec	r rec	both		
Assembly Hierarchical	30	-	-	30	-	-	-
Barman BDI	20	20	-	-	-	-	-
Blocksworld (GTOHP)	30	1	-	-	-	27	2
Blocksworld (HPDDL)	30	-	-	30	-	-	-
Childsnack	30	26	-	-	-	-	4
Depots	30	20	-	-	-	10	-
Elevator (L)	147	-	-	147	-	-	-
Entertainment	12	5	4	-	3	-	-
Factories	20	-	-	20	-	-	-
Freecell (L)	60	-	-	-	-	60	-
Hiking	30	-	-	-	26	-	4
Logistics (L)	80	-	-	80	-	-	-
Minecraft Player	20	-	-	4	-	-	16
Minecraft Regular	59	44	-	-	-	-	15
Monroe (FO)	20	-	-	-	-	20	-
Monroe (PO)	20	-	-	-	-	20	-
Multiarm Blocksworld	74	-	-	74	-	-	-
Robot	20	-	-	20	-	-	-
Rover (GTOHP)	30	2	-	-	-	28	-
Satellite (GTOHP)	20	-	-	-	-	20	-
Snake	20	-	-	20	-	-	-
Towers	20	-	-	20	-	-	-
Transport	40	-	40	-	-	-	-
Woodworking	30	30	-	-	-	-	-
	892	148	44	445	29	185	41

Table 1: Properties of the IPC 2020 benchmark set.

Though our approach is complete, our current implementation is not. The main problem is that FD uses graph search. For models translated using the approximation, we need a classical planner that will eventually return every solution to the problem (including certain loops that will not appear when using graph search). In FD, this cannot be realized easily. In our current implementation, FD only generates a single plan. When verification fails, we count the instance as unsolved. A complete implementation could e.g. be realized using Top-k planners (see e.g. Speck, Mattmüller, and Nebel (2020)). However, the incompleteness is not a problem on the 2020 IPC benchmark set. We have found exactly one instance where verification failed. This means that recursive loops change the state of the original HTN problem, such that the graph search does not prune the node.

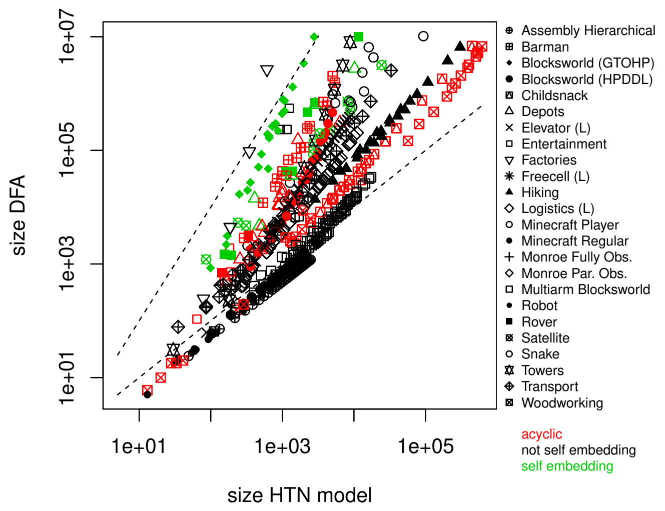

Figure 6: Comparison of the sizes of the HTN model and FA.

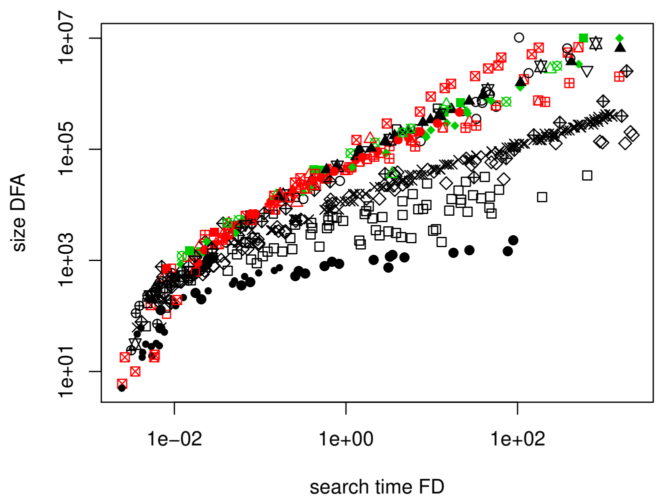

Figure 7: Size of the FA relative to the search time of FD.

		Absolute Coverage															Normalized Coverage				IPC Score			
	number instances	TOAD (EHC, G)	TOAD (G)	2STRIPS	**HyperTensioN**	**Lilotane**	PandaP (G, ADD)	PandaP (GA, ADD)	PandaP (G, FF)	PandaSAT	PandaP (GA, FF)	PandaP (G, LMC)	**PDDL4J**	PandaP (GA, LMC)	**SIADEX**	**PyHiPOP**	TOAD (EHC, G)	2STRIPS	**HyperTensioN**	**Lilotane**	TOAD (EHC, G)	2STRIPS	**HyperTensioN**	**Lilotane**
Assembly Hier.	30	**30**	**30**	25	3	5	**30**	3	11	5	4	6	2	2	0	1	**1.00**	0.83	0.10	0.17	**1.00**	0.44	0.08	0.12
Barman BDI	20	16	15	12	**20**	16	13	4	**20**	0	4	13	11	4	**20**	0	0.80	0.60	**1.00**	0.80	0.56	0.37	**1.00**	0.74
BW (GTOHP)	30	23	22	21	16	21	**29**	**29**	**29**	21	25	24	16	21	13	1	**0.77**	0.70	0.53	0.70	0.55	0.35	0.43	**0.61**
BW (HPDDL)	30	21	21	25	**30**	1	0	1	0	0	1	0	0	1	0	0	0.70	0.83	**1.00**	0.03	0.57	0.61	**0.89**	0.02
Childsnack	30	24	23	20	**30**	29	23	23	21	14	20	19	21	0	22	0	0.80	0.67	**1.00**	0.97	0.66	0.29	**1.00**	0.87
Depots	30	24	24	22	24	24	22	22	**27**	23	19	24	23	17	22	0	**0.80**	0.73	**0.80**	**0.80**	0.73	0.44	**0.76**	0.73
Elevator (L)	147	**147**	**147**	117	**147**	**147**	2	7	2	73	7	2	2	7	11	2	**1.00**	0.80	**1.00**	**1.00**	0.58	0.34	**1.00**	0.79
Entertainment	12	**12**	**12**	4	0	5	**12**	**12**	**12**	**12**	**12**	**12**	5	9	0	1	**1.00**	0.33	0.00	0.42	**0.86**	0.12	0.00	0.14
Factories	20	5	5	**6**	3	4	1	1	0	5	1	0	0	1	0	1	0.25	**0.30**	0.15	0.20	**0.20**	0.19	0.14	0.19
Freecell (L)	60	0	0	0	0	**9**	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	**0.15**	0.00	0.00	0.00	**0.05**
Hiking	30	23	23	24	**25**	22	24	23	23	17	12	7	17	1	0	0	0.77	0.80	**0.83**	0.73	0.53	0.45	**0.83**	0.60
Logistics (L)	80	49	50	**52**	22	43	0	4	1	26	4	1	0	4	0	0	0.61	**0.65**	0.28	0.54	**0.49**	0.32	0.26	0.32
Minecraft Pl.	20	1	1	4	**5**	1	4	1	4	0	0	1	1	0	3	0	0.05	0.20	**0.25**	0.05	0.02	0.04	**0.25**	0.03
Minecraft Reg.	59	41	41	55	**57**	30	44	44	43	22	43	41	23	40	35	0	0.69	0.93	**0.97**	0.51	0.52	0.54	**0.87**	0.34
Monroe (FO)	20	0	0	0	0	**20**	**20**	**20**	**20**	2	**20**	13	**20**	11	7	0	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	**0.78**
Monroe (PO)	20	0	0	0	0	**20**	11	12	12	2	12	9	1	7	0	0	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	**0.73**
Multiarm BW	74	**74**	**74**	**74**	8	4	29	43	4	0	5	3	0	4	1	0	**1.00**	**1.00**	0.11	0.05	**0.77**	0.63	0.11	0.03
Robot	20	**20**	**20**	**20**	**20**	11	1	9	2	0	8	1	6	8	0	1	**1.00**	**1.00**	**1.00**	0.55	0.95	0.84	**0.96**	0.52
Rover	30	9	9	11	**30**	21	27	26	22	9	12	18	27	7	**30**	6	0.30	0.37	**1.00**	0.70	0.25	0.23	**0.92**	0.54
Satellite	20	10	10	7	**20**	15	**20**	**20**	17	10	14	12	**20**	9	0	7	0.50	0.35	**1.00**	0.75	0.38	0.20	**1.00**	0.58
Snake	20	15	15	**20**	**20**	17	**20**	**20**	18	14	**20**	19	**20**	17	7	2	0.75	**1.00**	**1.00**	0.85	0.42	0.61	**1.00**	0.74
Towers	20	10	10	**17**	**17**	10	13	13	13	0	13	13	14	13	11	2	0.50	**0.85**	**0.85**	0.50	0.37	0.59	**0.78**	0.39
Transport	40	34	34	24	**40**	35	24	26	20	29	23	19	33	16	1	18	0.85	0.60	**1.00**	0.88	0.72	0.50	**1.00**	0.76
Woodworking	30	**30**	**30**	7	7	**30**	17	18	19	17	18	17	6	17	3	4	**1.00**	0.23	0.23	**1.00**	0.76	0.19	0.23	**0.98**
	892	**618**	616	567	544	540	386	381	340	301	297	274	268	216	186	46	**15.14**	13.78	14.10	14.34	11.89	8.29	**13.51**	11.60

Table 2: Three performance metrics for the planning systems. It starts with the absolute number of solved instances on the left. Since the domains do not have the same number of instances, we then included the percentage of solved instances per domain for selected systems. Last, their IPC score is given that combines coverage as well as the time needed to find the solutions. Please be aware that we fixed the ordering of the systems based on their absolute coverage. PandaP can be combined with arbitrary classical heuristics, we used Add (Bonet and Geffner 2001), FF (Hoffmann and Nebel 2001), and LM-Cut (Helmert and Domshlak 2009). Configurations labeled with *G* use GBFS, those marked *GA* use Greedy A* with weight 2.

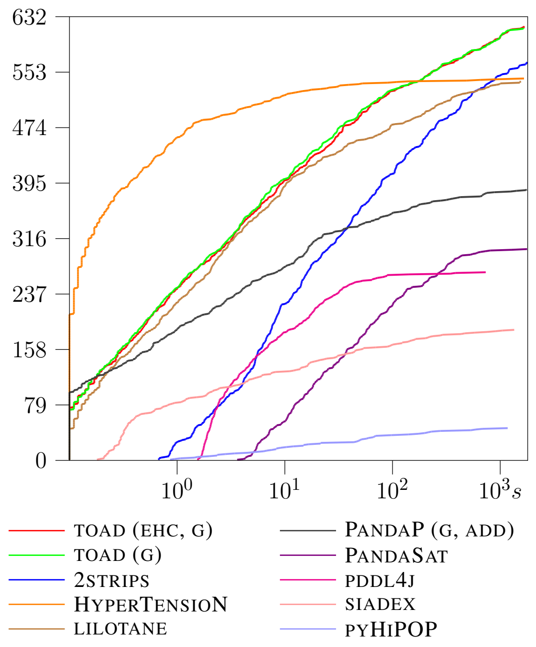

Figure 8: Solved instances (on the y axis) relative to the runtime in seconds (on the x axis, please be aware the log scale).

6 Evaluation

Analysis of the IPC 2020 Benchmark Set We first analyze the benchmark set of the IPC 2020 track on TOHTN planning with respect to the given properties. The results are given in Table 1. For each domain, it gives the total number of instances, followed by the number of instances that are non-recursive, (recursive but) non-self-embedding, and self-embedding. Since we test this properties on the ground model there are some instances where we do not know the properties (given in the last column) because grounding was not able given the memory limit. 147 problems from the benchmark set are non-recursive, from the recursive ones, 517 are not self-embedding. We can apply the exact translation to both of these classes, i.e. to 74% of the benchmark set. Most recursive but non-self-embedding problems are right recursive but there are also left recursive instances and those that are both left and right recursive (which is fine as long as there is not a *single* N_i that is both).

Wide parts of the benchmark set does not make use of the higher expressiveness of TOHTN planning compared to classical planning, they can as well be modeled with classical formalisms without changing the set of solutions.

Properties of the Generated Models Next we have a look at the properties of the generated models. Figure 6 compares the sizes of the HTN model and the FA. Our approach replaces the original tasks and methods with a set of actions, one action for each transition of the FA is introduced. Therefore we compare these numbers. Though we also add a huge number of state features representing FAs states, these are represented in a single SAS+ variable on the FD side. Each dot in the figure compares one instance. The lower diagonal line marks the border where the sizes are equal. The upper line the border where the FA is quadratic in the size of the HTN. It can be seen that most translations are of similar size compared to the input model. The green dots represent approximate models. These models tend to be bigger, while there is no real difference between the non-recursive (red) and the non-self-embedding (black) problems.

Figure 7 gives problem sizes relative to FD's search time. When comparing problems solved after a certain time, it can be seen that the non-recursive and approximated problems tend to be bigger than the recursive non-self-embedding problems. When doing the same plot with the HTN model's size on the y axis, the self-embedding problems tend to be on par with the non-self-embedding ones. I.e., (1) the non-recursive problems tend to be simpler to solve, and (2) the approximation makes them bigger, but not harder to solve.

Comparison to State of the Art Next we compare our system to other planners: the participants of the 2020 IPC: HYPERTENSION (Magnaguagno, Meneguzzi, and de Silva 2021), LILOTANE (Schreiber 2021), SIADEX (Fernandez-Olivares, Vellido, and Castillo 2021), PDDL4J (Pellier and Fiorino 2021), and PYHIPOP (Lesire and Albore 2021); PANDAP (Höller et al. 2018b) and PANDASAT (Behnke, Höller, and Biundo 2018) from the PANDA framework; and 2STRIPS (Alford et al. 2016).

Table 2 shows three performance metrics, first the absolute coverage. Since the domains have a different number of instances, we included (for selected systems) the percentage of solved problems per domain. Last, it includes the IPC score. TOAD reaches the highest coverage, followed by 2STRIPS and the IPC systems (in the same order as in the IPC). Our system solves 51 instances more than 2STRIPS, and 74 more instances than the best IPC planner. When we consider the percentage of solved problems, our systems still beat the others. However, the second best system is now LILOTANE (the runner up from the IPC), followed by HYPERTENSION (the winner of the IPC), and 2STRIPS.

When we look at the IPC score, we see that our system is on the second place, beaten by HYPERTENSION. The reason for this can be seen in Figure 8. It shows the accumulated number of solved instances (on the y axis) over time (on the x axis). HYPERTENSION solves a majority of its instances within the first second, and nearly all of them in the first 10 seconds. Our system needs approximately 100 seconds to overtake HYPERTENSION. This decreases the IPC score of our system. The given behavior of HYPERTENSION is not surprising given that it uses a lifted depth first search.

7 Conclusion

We presented a new approach to solve TOHTN problems by translating them to classical planning problems. Instead of bounding the problems like existing translations to classical problems or SAT, we overapproximate the set of solutions and verify solutions afterwards. However, for the vast majority of problems in the 2020 IPC benchmark set, we can even use an exact translation. Our empirical evaluation shows that our translation combined with the FD planning system outperforms the planners from the 2020 IPC as well as several other HTN planners in terms of coverage.

Acknowledgments

I want to thank Florian Pommerening and Gregor Behnke for the helpful discussions on the paper.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 232722074 – SFB 1102 / Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 232722074 – SFB 1102.

References

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. W. 2016. Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems. In *Proc. of ICAPS*, 20–28. AAAI Press.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *Proc. of IJCAI*, 1629–1634.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting Procedural Domain Control Knowledge in State-of-the-Art Planners. In *Proc. of ICAPS*, 26–33. AAAI.

Behnke, G.; Bercher, P.; Kraus, M.; Schiller, M. R. G.; Mickeleit, K.; Häge, T.; Dorna, M.; Dambier, M.; Manstetten, D.; Minker, W.; Glimm, B.; and Biundo, S. 2020a. New Developments for Robert – Assisting Novice Users Even Better in DIY Projects. In *Proc. of ICAPS*, 343–347. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning Through SAT. In *Proc. of AAAI*, 6110–6118. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proc. of AAAI*, 7520–7529. AAAI Press.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020b. On Succinct Groundings of HTN Planning Problems. In *Proc. of AAAI*, 9775–9784. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proc. of IJCAI*, 6267–6275. IJCAI.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2): 5–33.

Chrpa, L.; and Barták, R. 2016. Guiding Planning Engines by Transition-Based Domain Control Knowledge. In *Proc. of KR*, 545–548. AAAI Press.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence (AMAI)* 18(1): 69–93.

Fernandez-Olivares, J.; Vellido, I.; and Castillo, L. 2021. Addressing HTN Planning with Blind Depth First Search. In *10th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proc. of IJCAI*, 1955–1961. AAAI.

Goldman, R. P. 2009. A Semantics for HTN Methods. In *Proc. of ICAPS*, 146–153. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. of ICAPS*, 162–169. AAAI Press.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14: 253–302.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of ECAI*, 447–452. IOS Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *Proc. of ICAPS*, 158–165. AAAI Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018a. Plan and Goal Recognition as HTN Planning. In *Proc. of ICTAI*, 466–473. IEEE Press.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020a. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proc. of AAAI*, 9883–9891. AAAI Press.

Höller, D.; Bercher, P.; and Behnke, G. 2020. Delete- and Ordering-Relaxation Heuristics for HTN Planning. In *Proc. of IJCAI*, 4076–4083. IJCAI.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018b. A Generic Method to Guide HTN Progression Search with Classical Heuristics. In *Proc. of ICAPS*, 114–122. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2019. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *Proc. of IJCAI*, 6171–6175. IJCAI.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020b. HTN Planning as Heuristic Progression Search. *Journal of Artificial Intelligence Research* 67: 835–880.

Köhn, A.; Wichlacz, J.; Torralba, Á.; Höller, D.; Hoffmann, J.; and Koller, A. 2020. Generating Instructions at Different Levels of Abstraction. In *Proc. of COLING*, 2802–2813. International Committee on Computational Linguistics.

Lesire, C.; and Albore, A. 2021. PYHIPOP – Hierarchical Partial-Order Planner. In *10th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Magnaguagno, M. C.; Meneguzzi, F.; and de Silva, L. 2021. HyperTensioN – A three-stage compiler for planning. In *10th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research* 20: 379–404.

Nederhof, M.-J. 2000a. Practical experiments with regular approximation of context-free languages. *Computational Linguistics* 26(1): 17–44.

Nederhof, M.-J. 2000b. Regular approximation of CFLs: A grammatical view. In *Advances in Probabilistic and other Parsing Technologies*, chapter 12, 221–241. Kluwer Academic Publishers.

Pellier, D.; and Fiorino, H. 2021. Totally and Partially Ordered Hierarchical Planners in PDDL4J Library. In *10th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Schreiber, D. 2021. Lifted Logic for Task Networks: TOHTN Planner Lilotane Entering IPC 2020. In *10th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Schreiber, D.; Pellier, D.; Fiorino, H.; and Balyo, T. 2019. Tree-REX: SAT-Based Tree Exploration for Efficient and High-Quality HTN Planning. In *Proc. of ICAPS*, 382–390. AAAI Press.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. of AAAI*, 9967–9974. AAAI Press.