On the Verification of Totally-Ordered HTN Plans

Roman Barták,[1] **Simona Ondrčková,** [1] **Gregor Behnke,** [2] **Pascal Bercher** [3]

[1] Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
[2] University of Freiburg, Faculty of Engineering, Freiburg, Germany
[3] The Australian National University, College of Engineering & Computer Science, Canberra, Australia
{bartak,ondrckova}@ktiml.mff.cuni.cz, behnkeg@informatik.uni-freiburg.de, pascal.bercher@anu.edu.au

Abstract

Verifying HTN plans is an intractable problem with two existing approaches to solve the problem. One technique is based on compilation to SAT. Another method is using parsing, and it is currently the fastest technique for verifying HTN plans. In this paper, we propose an extension of the parsing-based approach to verify totally-ordered HTN plans more efficiently. This problem is known to be tractable if no state constraints are included, and we show theoretically and empirically that the modified parsing approach achieves better performance than the currently fastest HTN plan verifier when applied to totally-ordered HTN plans.

Introduction

Plan verification is about finding if a given action sequence forms a correct plan according to a given planning domain model. For classical plans, the verification problem consists of checking if the action sequence is executable starting with the initial state and checking if the goal condition is satisfied in the final state (Howey and Long 2003). For hierarchical plans, plan verification additionally requires that the action sequence can be obtained by decomposition of some task. A specific root task, which decomposes to the action sequence, might also be given to describe the goal task.

There exist two approaches to hierarchical plan verification. One uses a translation of the verification problem into a Boolean satisfiability problem (Behnke, Höller, and Biundo 2017). The second uses parsing and it supports state constraints (Barták, Maillard, and Cardoso 2018; Barták et al. 2020). The hierarchical planning domain model can be seen as a formal grammar (Höller et al. 2014; Höller et al. 2016; Barták and Maillard 2017) and the plan verification problem is then similar to checking if a word (action sequence) belongs to the language generated by the grammar, which can be done by parsing. Parsing does not require information about the goal task – the method finds any task that decomposes to the action sequence, which makes it appropriate also for plan and task recognition (Vilain 1990).

The parsing-based approach seems to be significantly faster than the SAT-based approach (Barták et al. 2020). Nevertheless, both approaches struggle from the combinatorial explosion and, depending on the domain; they can verify plans of lengths up to a few dozens of actions. This is not surprising as the problem of verifying hierarchical plans is NP-hard (Behnke, Höller, and Biundo 2015; Bercher et al. 2016) and hence computationally expensive. This holds for general hierarchical plans with task interleaving and the partial order of tasks. However, as can be seen from the International Planning Competition 2020 on HTN planning, many domain models contain totally-ordered tasks. Further, there is a significant body of research dedicated to totally-ordered HTN planning in particular (Olz, Biundo, and Bercher 2021; Behnke and Speck 2021; Behnke 2021; Lin and Bercher 2021; Schreiber et al. 2019; Behnke, Höller, and Biundo 2018; Alford, Kuter, and Nau 2009; Marthi, Russell, and Wolfe 2007; Nau et al. 1999). Plan verification for a totally-ordered problem without state constraints is known to be tractable (Behnke, Höller, and Biundo 2015). Nevertheless, no hierarchical plan verifier exploits this theoretical result, no verifier specializes in exploiting the total order, and no generalisation to problems with state constraints exists.

We propose an extension of the parsing-based verification algorithm (Barták et al. 2020) to work faster for totally-ordered domain models. While the CYK algorithm (Sakai 1962) for parsing context-free grammars appears to be applicable here at first glance, this is not the case. Totally-ordered models can contain state constraints, which cannot, to our current knowledge, be compiled or handled by the CYK algorithm. Our primary modification is in handling precedence relations in the totally-ordered setting. The extended algorithm still works for arbitrary partially ordered hierarchical plans. It detects if the model uses totally-ordered tasks, and then uses a more strict formulation of precedence constraints, which decreases the number of generated tasks significantly. The algorithm works in a bottom-up fashion starting with a given action sequence $\bar{a}$. It terminates once a compound task is found that can be decomposed into $\bar{a}$. Apart from other work on plan verification, our approach is loosely related to another that aims at computing abstract plans that are maximally abstract while still allowing to generate a non-redundant plan (de Silva, Padgham, and Sardina 2019). The proposed algorithm also performs a bottom-up approach, though it requires a specific decomposition rather than the entire model.

HTN Plan Verification by Parsing

We use a standard STRIPS formalization (Fikes and Nilsson 1971). Let P be a set of propositions describing prop-

erties of world states. Then, a world state is modeled as a set $S \subseteq P$ of propositions that are true in that state (every other proposition is false). Each action a is modeled by three sets of propositions $(\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a))$, where $\texttt{pre}(a), \texttt{eff}^+(a), \texttt{eff}^-(a) \subseteq P$ and $\texttt{eff}^+(a) \cap \texttt{eff}^-(a) = \emptyset$. The set $\texttt{pre}(a)$ describes positive preconditions of action a. These propositions must be true right before the action a. Action a is applicable to state S iff $\texttt{pre}(a) \subseteq S$. Sets $\texttt{eff}^+(a)$ and $\texttt{eff}^-(a)$ describe the positive and negative effects of action a. These propositions will become true or false in the state right after executing the action a. If an action a is applicable to state S then the state right after the action a is:

$$\gamma(S, a) = (S \setminus \texttt{eff}^-(a)) \cup \texttt{eff}^+(a).$$

$\gamma(S, a)$ is undefined if action a is not applicable to state S. We say that an action sequence $(a_1, \ldots, a_n)$ is *executable* with respect to a given initial state S_0 if the precondition of each action is satisfied in the state right before it:

$$\texttt{pre}(a_i) \subseteq \gamma(\gamma(\ldots\gamma(S_0, a_1), \ldots), a_{i-1}).$$

Hierarchical Task Network Planning (Erol, Hendler, and Nau 1996) was proposed as a planning framework that includes control knowledge as recipes for solving specific tasks. The recipe is modeled using a task network – a set of sub-tasks to solve the task and a set (a conjunction) of constraints between the sub-tasks. Let T be a compound task and $(\{T_1, ..., T_k\}, C)$ be a task network, where C are its constraints (see later). We can describe the decomposition method as a rewriting rule saying that T decomposes to sub-tasks $T_1, ..., T_k$ under the constraints C:

$$T \rightarrow T_1, ..., T_k \ \ [C]$$

The order of sub-tasks in the rule does not matter (opposite to rewriting rules in grammars) as the precedence constraints in C explicitly describe the order. If the tasks $T_1, ..., T_k$ in each method are totally ordered, then we speak about a *totally-ordered HTN model*.

HTN planning problems are specified by an initial state S_0 and an initial task representing the goal. This goal task needs to be decomposed via decomposition methods until a set of primitive tasks – actions – is obtained. These actions must be totally ordered and satisfy all the constraints obtained during decompositions. The obtained plan $(a_1, \ldots, a_n)$ must be executable with respect to S_0. The state right after the action a_i is denoted S_i. We denote the set of actions to which a task T decomposes as $act(T)$. If U is a set of tasks, we define $act(U) = \cup_{T \in U} act(T)$. The index of the first action in the decomposition of T is denoted $start(T)$, that is, $start(T) = min\{i | a_i \in act(T)\}$. Similarly, $end(T)$ means the index of the last action in the decomposition of T, that is, $end(T) = max\{i | a_i \in act(T)\}$.

The decomposition constraints for a method $T \rightarrow T_1, ..., T_k$ can be of the following three types, where the first is also known as an ordering constraint and the latter two are essentially state constraints ($U, V, \{t_1, t_2\} \subseteq \{T_1, ..., T_k\}$):

- $t_1 \prec t_2$: a precedence constraint meaning that in every plan the last action obtained from task t_1 is before the first action obtained from task t_2, $end(t_1) < start(t_2)$,
- *before*(p, U): a precondition constraint meaning that in every plan the proposition p holds in the state right before the first action obtained from tasks U, $p \in S_{start(U)-1}$,
- *between*(U, p, V): a prevailing constraint meaning that in every plan the proposition p holds in all the states between the last action obtained from tasks U and the first action obtained from tasks V, $\forall i \in \{end(U), \ldots, start(V) - 1\}, p \in S_i$.

The *HTN plan verification problem* is formulated as follows: Given a sequence of actions $(a_1, a_2, \ldots, a_n)$ and an initial state S_0, is the sequence of actions executable with respect to S_0 and obtained from some compound task?

Algorithm 1 presents the recent parsing-based approach to HTN plan verification (Barták et al. 2020) extended with the check of total-order constraints at line 13 (see the next section). The set $\prec$ represents the precedence constraints of the method, bef is the set of $before$ constraints, and btw is the set of $between$ constraints. Executability of the action sequence is verified at lines 2-5. The `while` loop (lines 7-26) groups actions/tasks into compound tasks by using the methods from the model until it finds a task T_0 such that $act(T_0) = \{a_1, a_2, \ldots, a_n\}$ (line 26, the plan is valid) or it constructs all possible tasks that decompose to a subset of actions in the plan (line 27, the plan is invalid). The sets $act(T)$ are represented using Boolean vectors I ($I(j) = 1 \Leftrightarrow a_j \in act(T)$). These vectors are used to check that each action is generated from one task only (line 19). Indexes b_j and e_j for task T_j describe values $start(T_j)$ and $end(T_j)$ respectively. They are used when checking the decomposition constraints.

Totally-Ordered HTNs

The parsing-based verification algorithm may generate an exponential number of pairs $(T, act(T))$, where T is a task and $act(T)$ is a subset of actions from the plan that can be generated from the task T. This is because actions from different tasks may interleave in the plan, and hence we must assume subsets $act(T)$ of actions from the plan when composing the tasks T. There is an exponential number of such sets with respect to the length of the plan. However, when the domain model is totally ordered, then the sets $act(T)$ form contiguous sub-sequences of actions (Figure 1).

Proposition 1. *For a totally ordered HTN domain model, each task decomposes to a contiguous sub-sequence of actions in the plan.*

Proof. Assume a pair of different tasks T and T' used in the decomposition of some goal task T_g to a sequence of actions such that T and T' are not descendants of each other. There must exists a common ancestor task T_a for tasks T and T' in the decomposition tree and a method $T_a \rightarrow T_1, ..., T_k \ \ [C]$ used for the decomposition. Let the task T be obtained from the sub-task T_i and T' be obtained from the sub-task T_j. As the domain model is totally-ordered, without loss of generality, we may assume that $T_i \prec T_j$ and hence $end(T_i) < start(T_j)$. As T is a sub-task of T_i, we know $end(T) \leq end(T_i)$ and similarly $start(T_j) \leq start(T')$. Together we

get $end(T) < start(T')$. Hence for any pair of non descendant tasks T and T', it holds either $end(T) < start(T')$ or $end(T') < start(T)$, which means that the tasks do not interleave in the plan. □

We can exploit this property when verifying plans for totally-ordered domain models as follows. Assume a decomposition method $T \rightarrow T_1, ..., T_k\ [C]$ in a totally-ordered domain model. Then it holds $\forall i \in \{1, \ldots, k-1\} : T_i \prec T_{i+1}$. We call these precedence constraints *direct precedences* to distinguish them from classical precedence relations. Note that it is easy to detect automatically, if the domain model is totally ordered, for example, by using a transitive closure of precedence relations in the decomposition methods and verifying that sub-tasks in the method are totally ordered. The direct precedence relation $T_i \prec T_{i+1}$ means that the last action of task T_i is right before the first action of task T_{i+1}. This is a consequence of Proposition 1. Task T decomposes to a contiguous action sequence P. Each of its sub-tasks T_i also decomposes to a contiguous action sequence and these sub-sequences are ordered as $end(T_i) < start(T_{i+1})$. Together these sub-sequences must form the sequence P without any gap. Hence, the direct precedence relation imposes a more strict constraint

$$end(T_i) + 1 = start(T_{i+1}). \quad (1)$$

Note that the above claim also holds in the reverse order. Suppose we impose the above ordering constraint (1) for direct precedence relations in all decomposition methods. In that case, the tasks decompose to contiguous sequences of actions as no action can be inserted between any pair of directly following tasks.

The extended HTN plan verification algorithm (Algorithm 1) checks the direct precedence constraints for totally-ordered domain models at line 13. This extension gives the theoretical guarantee on the number of generated tasks.

Proposition 2. *Let t be the number of tasks in the totally-ordered HTN domain model and n be the number of actions in a plan. Then the extended HTN plan verification algorithm generates at most $O(t \times n^2)$ different pairs $(T, act(T))$.*

Proof. For totally ordered domain models, the sets $act(T)$ form contiguous sub-sequences of the plan. These sub-sequences are identified by the first and the last actions in the sequence, and hence there are at most $O(n^2)$ such sets. The same set of actions may be generated from different tasks; hence the maximal number of different pairs $(T, act(T))$ that the parsing-based verification algorithm may generate is $O(t \times n^2)$. □

Note that if the original verification algorithm is applied to totally-ordered domain models, then it may still generate an exponential number of pairs $(T, act(T))$ because the algorithm allows sets $act(T)$ to be arbitrary subsets of actions in the plan. The experimental study confirms this.

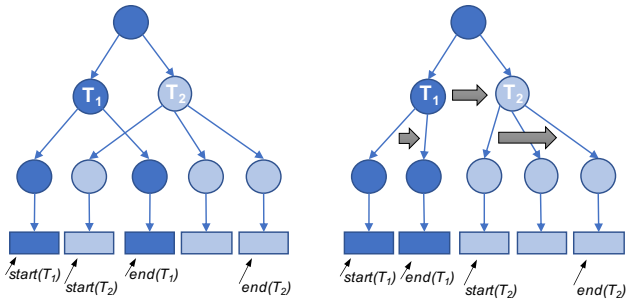

Figure 1: Task interleaving (left) vs. totally ordered (right).

Data: a plan $\mathbf{P} = (a_1, ..., a_n)$, an initial state S_0, and a set of decomposition methods (domain model); $TO = true$ if the domain is totally ordered,
Result: `true` if the plan can be derived from some compound task, `false` otherwise

1 **Function** VERIFYPLAN
2 **for** *i = 1* **to** *n* **do**
3 **if** $\neg(\text{pre}(a_i) \subseteq S_{i-1})$ **then**
4 **return false**
5 $S_i = (S_{i-1} \setminus \text{eff}^-(a_i)) \cup \text{eff}^+(a_i)$
6 $\mathbf{sp} \leftarrow \emptyset$; new $\leftarrow \{(A_i, i, i, I_i)\ |i \in 1..n\}$
  Data: A_i is a primitive task corresponding to action a_i, I_i is a Boolean vector of size n, such that $\forall i \in 1..n, I_i(i) = 1$, $\forall j \neq i, I_i(j) = 0$
7 **while** new $\neq \emptyset$ **do**
8 $\mathbf{sp} \leftarrow \mathbf{sp} \cup$ new; new $\leftarrow \emptyset$
9 **foreach** *decomposition method R of the form* $T_0 \rightarrow T_1, ..., T_k\ [\prec, \text{bef}, \text{btw}]$ *such that* $\{(T_j, b_j, e_j, I_j)|j \in 1..k\} \subseteq \mathbf{sp}$ **do**
10 **if** $\exists (i,j) \in \prec : \neg(e_i < b_j)$ **then**
11 **continue with the next method**
12 **if** $TO \wedge \exists i : \neg(e_i + 1 = b_{i+1})$ **then**
13 **continue with the next method**
14 $b_0 \leftarrow \min\{b_j|j \in 1..k\}$
15 $e_0 \leftarrow \max\{e_j|j \in 1..k\}$
16 **for** *i = 1* **to** *n* **do**
17 $I_0(i) \leftarrow \sum_{j=1}^{k} I_j(i)$;
18 **if** $I_0(i) > 1$ **then**
19 **continue with the next method**
20 **if** $\exists (p, U) \in \text{bef} : p \notin S_{\min\{b_j|j \in U\}-1}$ **then**
21 **continue with the next method**
22 **if** $\exists (U, p, V) \in \text{btw}\ \exists i \in \max\{e_j|j \in U\}, \ldots, \min\{b_j|j \in V\} - 1 : p \notin S_i$ **then**
23 **continue with the next method**
24 new $\leftarrow$ new $\cup \{(T_0, b_0, e_0, I_0)\}$
25 **if** $\forall k, I_0(k) = 1$ **then**
26 **return true**
27 **return false**

Algorithm 1: Parsing-based HTN plan verification

Empirical Evaluation

We compared the recent HTN plan verification algorithm (Barták et al. 2020) with its extended version that detects totally-ordered domain models and imposes constraints (1) to check the direct precedence constraints used in decomposition methods. Compared to previous evaluations, we have significantly increased the number of instances we consider. The International Planning Competition (IPC) 2020 has released an extensive set of plans[1] that were generated by the planners in the IPC on the IPC domains[2]. We are using the set of totally-ordered plans provided by the IPC, that is, all plans in our evaluation are totally-ordered. This set contains 10963 plans with an average length of 239 actions and a maximum length of 131071 actions.

Both the original verifier (Barták et al. 2020) and the modifications presented in this paper were implemented in C# 7 (from .NET 4.7). For running the program, we used mono in version 6.8.0.105 on a singularity container based on Ubuntu 20.10. We ran all experiments on an Intel Xeon Gold 6242 CPU (2.80GHz) with 5GB of RAM and a timeout of 10 minutes. The memory limit was never reached.

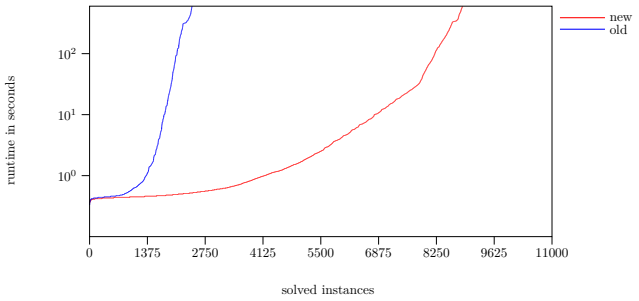

Figure 2: The number of solved problems per time.

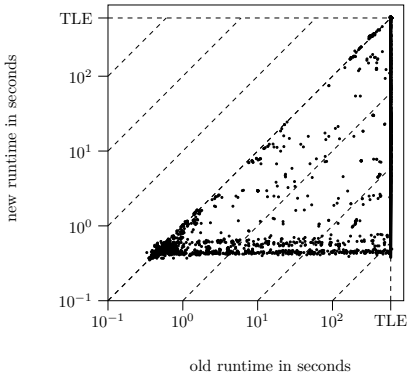

Figure 3: Direct comparison of runtimes.

The summary results are presented in Figure 2 showing the number of solved instances within a given time. The new approach solves a significantly larger number of instances (8870) than the original approach (2443). Any instance solved by the original approach was also solved by the new approach, while the new approach solved 6427 instances more. Figure 3 presents the direct comparison of both techniques using the same data. Each point represents one of the 2443 problem instances solved by both approaches. The runtimes of the algorithms define the coordinates of the point. Of these 2443 instances, the old approach is faster in 362 instances. Of these 362, only 159 have a runtime of more than one second. For these 362 instances, the old algorithm is faster than the new one by more than 10% in only 24 instances and at the most only 25% faster. The minor overhead of the new algorithm seems not to incur a significant disadvantage. For 210 instances, the runtime is identical, and for the remaining 1871 instances solved by both verifiers, the runtime of the new one is faster. The reduction in runtime on these 1871 instances is on average 46.36% with a maximum of 99.93%.

[1]https://github.com/panda-planner-dev/ipc-2020-plans

[2]https://github.com/panda-planner-dev/ipc2020-domains

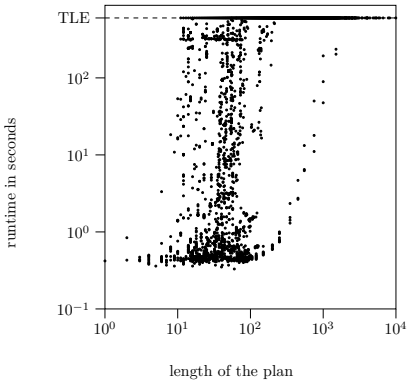

Figure 4: Runtime of the original algorithm as a function of plan length (omitting plans with more than 10.000 actions).

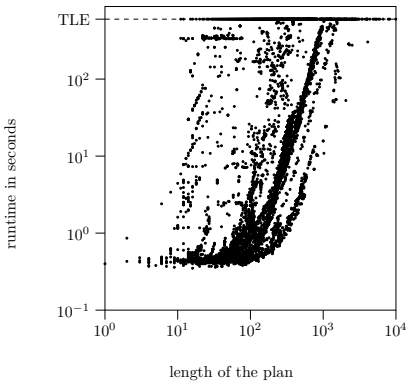

Figure 5: Runtime of the extended algorithm as a function of plan length (omitting plans with more than 10.000 actions).

Figures 4 and 5 show the dependence of runtime on plan length for the original and extended algorithm, respectively. Again, it is clearly visible that the new method solves a larger number of instances. The new approach can verify about one order of magnitude longer plans than the original algorithm. The longest verified plan for the old technique has 1500 actions, while for the new one has 4095 actions.

Conclusions

We proposed extending the HTN plan verification algorithm to impose a more strict constraint describing direct precedence relations for totally-ordered models. The effect of this modification on the runtime of the algorithm is dramatic. The new algorithm verifies a much larger number of problem instances and also longer plans. As totally-ordered HTN domain models are frequent in practical applications, the method brings automated HTN plan verification closer to practical applicability on non-trivial plans and domains.

Acknowledgments. Research was supported by the joint Czech-German project registered under the number 21-13882J by the Czech Science Foundation (GAČR) and BE 7458/1-1 by Deutsche Forschungsgemeinschaft (DFG). S. Ondrčková is supported by SVV project number 260 575.

References

Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *Proc. of the 21st Int. Joint Conf. on AI (IJCAI 2009)*, 1629–1634. AAAI Press.

Barták, R.; and Maillard, A. 2017. Attribute Grammars with Set Attributes and Global Constraints As a Unifying Framework for Planning Domain Models. In *Proc. of the 19th Int. Symposium on Principles and Practice of Declarative Programming (PPDP 2017)*, 39–48. ACM.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of Hierarchical Plans via Parsing of Attribute Grammars. In *Proc. of the 28th Int. Conf. on Automated Planning and Scheduling (ICAPS 2018)*, 11–19. AAAI Press.

Barták, R.; Ondrčková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A Novel Parsing-based Approach for Verification of Hierarchical Plans. In *Proc. of the 32nd Int. Conf. on Tools with AI (ICTAI 2020)*, 118–125. IEEE.

Behnke, G. 2021. Block Compression and Invariant Pruning for SAT-based Totally-Ordered HTN Planning. In *Proc. of the 31st Int. Conf. on Automated Planning and Scheduling (ICAPS 2021)*, 25–35. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition. In *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS 2015)*, 25–33. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This Is a Solution! (... But Is It Though?) - Verifying Solutions of Hierarchical Planning Problems. In *Proc. of the 27th Int. Conf. on Automated Planning and Scheduling (ICAPS 2017)*, 20–28. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning through SAT. In *Proc. of the 32nd AAAI Conf. on AI (AAAI 2018)*, 6110–6118. AAAI Press.

Behnke, G.; and Speck, D. 2021. Symbolic Search for Optimal Total-Order HTN Planning. In *Proc. of the 35th AAAI Conf. on AI (AAAI 2021)*, 11744–11754. AAAI Press.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo, S. 2016. More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks. In *Proc. of the 22nd European Conf. on AI (ECAI)*, 225–233. IOS Press.

de Silva, L.; Padgham, L.; and Sardina, S. 2019. HTN-Like Solutions for Classical Planning Problems: An Application to BDI Agent Systems. *Theoretical Computer Science* 763: 12–37.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Annals of Mathematics and AI* 18(1): 69–93.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. of the 2nd Int. Joint Conf. on AI (IJCAI 1971)*, 608–620.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of the 21st European Conf. on AI (ECAI 2014)*, 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 the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS 2016)*, 158–165. AAAI Press.

Howey, R.; and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the Int. Planning Competition. In *Proc. of the ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*.

Lin, S.; and Bercher, P. 2021. Change the World – How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *Proc. of the 30th Int. Joint Conf. on AI (IJCAI 2021)*. IJCAI.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Angelic Semantics for High-Level Actions. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling (ICAPS 2007)*, 232–239. AAAI Press.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proc. of the 16th Int. Joint Conf. on AI (IJCAI 1999)*, 968–973.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks – A Complexity Analysis. In *Proc. of the 35th AAAI Conf. on AI (AAAI 2021)*, 11903–11912. AAAI Press.

Sakai, I. 1962. Syntax in universal translation. In *Proc. of the 1961 Int. Conf. on Machine Translation of Languages and Applied Language Analysis*, 593–608.

Schreiber, D.; Balyo, T.; Pellier, D.; and Fiorino, H. 2019. Tree-REX: SAT-based Tree Exploration for Efficient and High-Quality HTN Planning. In *Proc. of the 29th Int. Conf. on Automated Planning and Scheduling (ICAPS 2019)*, 382–390. AAAI Press.

Vilain, M. 1990. Getting Serious About Parsing Plans: A Grammatical Analysis of Plan Recognition. In *Proc. of the 8th National Conf. on AI (AAAI)*, 190–197. AAAI Press.