Using Earley Parser for Verification of Totally Ordered Hierarchical Plans

Kristýna Pantůčková, Simona Ondrčková, Roman Barták
Charles University, Faculty of Mathematics and Physics
Prague, Czech Republic
{pantuckova, ondrckova, bartak}@ktiml.mff.cuni.cz

Abstract

Hierarchical planning extends classical planning by capturing the hierarchical structure of tasks via decomposition of tasks to subtasks. Hierarchical plan verification is the problem of determining whether a given plan is valid according to that structure. As decomposition trees in totally ordered hierarchical planning domains resemble parsing trees of context-free grammars, one may exploit the techniques for context-free grammars also for hierarchical plans. In this paper, we study how the Earley parser, proposed for checking whether a given word belongs to a language defined by context-free grammar, can be extended to verification of totally ordered hierarchical plans.

Introduction

Hierarchical planning (Erol, Hendler, and Nau 1996) extends classical planning by a task hierarchy describing how tasks decompose to subtasks until primitive tasks – actions – are obtained. A hierarchical planning domain model defines possible task decompositions via decomposition rules. The planning problem is given by a goal task and an initial state and the problem is to find the decomposition of the goal tasks into a sequence of actions executable in the initial state. The plan verification problem is somehow reverse to the planning problem – the input consists of an action sequence and an initial state, and the verification problem is to check that the action sequence is a valid hierarchical plan, that is, it can be obtained by decomposition of some goal task and it is executable in the initial state. The goal task may or may not be given as input.

Plan verification is not just an academic problem, it can be used, for example, to verify plans in planning competitions, and it has various practical applications. For example, users need to check that a given action sequence complies with the rules describing the task, which the action sequence solves (in medical surgeries, manufacturing processes, handling customers, etc.).

Verifying, that an action sequence is executable in a given state, is straightforward – a well known system VAL is used there for a long time (Howey and Long 2003). The focus of this paper is on verifying hierarchical plans, in particular reconstructing the decomposition structure, which is a much harder problem. The first hierarchical plan verification approach was based on compilation to Boolean satisfiability problem (Behnke, Höller, and Biundo 2017). Then approaches based on parsing became more widespread (Barták, Maillard, and Cardoso 2018; Lin et al. 2023; Ondrčková et al. 2023) and also compilation of hierarchical plan verification problem to hierarchical planning has been suggested (Höller et al. 2022). Existing parsing-based approaches to hierarchical plan verification use a bottom-up approach; they start with the action sequence and build a decomposition tree towards the goal task. The planning-based approach uses a top-down method, going from the goal task towards the plan, and may exploit various planning heuristics. Using heuristics seems more complicated for bottom-up approaches (Ondrčková et al. 2022).

In this paper, we present another parsing-based approach to hierarchical plan verification, which is based on the Earley parser (Earley 1970). The Earley parser was originally proposed for context-free grammars; therefore, our approach is limited to verification of totally ordered plans. In totally ordered planning domains, every decomposition rule decomposes a task into a totally ordered sequence of subtasks. Nevertheless, many planning domains can be naturally defined as totally ordered domains. In the International Planning Competition (IPC) 2020, 24 of 33 planning domains were totally ordered. Earley parser has been already used for hierarchical plan recognition, where it outperformed the existing parsing-based approach (Pantůčková and Barták 2023). Moreover, the Earley parser is based on top-down parsing, progressing from possible goal tasks downwards to the actions from the input plan, which could simplify introduction of planning heuristics to it.

The paper is organized as follows: Firstly, we provide background on hierarchical plan verification. Secondly, we describe the related work. Then, we describe our approach to plan verification. Finally, we present the results of empirical comparison of our approach and other state of the art hierarchical plan verifiers. We also compare the performance of the Earley parser on grounded and lifted problems, showing that grounding improves the performance of verification of longer plans. The grounded Earley parser shows a performance comparable to the other parsing-based approaches.

Background on hierarchical plan verification

Hierarchical planning domains define two types of tasks: abstract (compound) tasks, which decompose into simpler tasks, and primitive tasks – actions. Similarly to classical planning, actions are defined by preconditions (atomic propositions that must be true before the action is executed) and effects (atomic propositions that will become true or false after the action is executed). Possible task decompositions are defined by decomposition rules. A rule $T \rightarrow T_1, ..., T_n\ [C]$ describes decomposition of task T into subtasks $T_1, ..., T_n$, where C is a set of rule constraints.

Hierarchical planning domains are often based on the formalism of hierarchical task networks (HTN), which define three types of rule constraints:

- before-constraints: *before*(T', p), s.t. $T' \subseteq \{T_1, ..., T_n\}$ and the action sequence into which the tasks in T decompose starts with the action a, indicates that the proposition p must hold in the state where a is executed;
- between-constraints: *between*(T', T'', p), s.t. $T', T'' \subseteq \{T_1, ..., T_n\}$ and the sequence into which the tasks in T' decompose ends with the action a and T'' decomposes into a sequence of actions starting with b, indicates that p must hold in all states between the execution of a and b;
- ordering-constraints: $T_i \prec T_j$, where the task T_i decomposes into a sequence of actions ending with a and T_j decomposes into a sequence of actions starting with b, enforces that a must be executed before b.

In totally ordered hierarchical planning, the subtasks in each rule are totally ordered, i.e., $T_i \prec T_j$ for all $i < j$.

Figure 1 shows an example of task decomposition in a classical planning domain Blocksworld, where towers of blocks are being constructed. Let *a*, *b* and *c* be three blocks. Our goal is to put the block *b* on the block *a*. As there already is another block (*c*) on the block *a*, we firstly need to remove the blocks above the block *a*. Therefore, the root task *put_on(b,a)* decomposes into two tasks: the abstract task *clear(a)* and the action *stack(b,a)*. The task *clear(a)* decomposes into the action *unstack(c,a)*. The resulting plan is a sequence of two actions: *unstack(c,a), stack(b,a)*.

As an input of an HTN plan verification problem, a verifier receives the plan *unstack(c,a), stack(b,a)*, the description of the planning domain and possibly the goal task *put_on(b, a)*. The purpose of the verifier is to decide whether the given goal task decomposes into the given plan (or whether there is any goal task that decomposes into the plan).

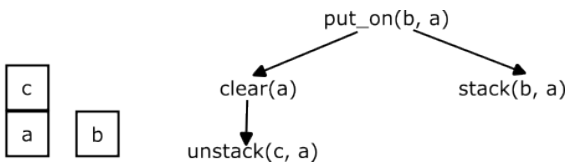

Figure 1: Example of a hierarchical task decomposition.

Related works

The first approach to solve hierarchical plan verification problems was based on encoding the problem into a Boolean satisfiability problem (Behnke, Höller, and Biundo 2017). This encoding does not support before and between constraints and it is outperformed by recent techniques.

Another compilation-based approach encodes the verification problem as a planning problem (Höller et al. 2022). The authors present an encoding, which translates a plan verification into a hierarchical planning problem, which is then solved by an HTN planner. If a planner finds a plan for the new planning problem, the original plan is valid.

The first approach that supports all types of rule constraints and also partial order of tasks (task interleaving) was based on parsing (Barták, Maillard, and Cardoso 2018). This idea motivated other parsing-based techniques with significantly better performance (Barták et al. 2020). The well-known Cocke–Younger–Kasami algorithm (CYK), originally proposed for parsing of context-free grammars, has been modified to hierarchical plan verification (Lin et al. 2023). As an input, this approach requires a grounded problem (task attributes are substituted by constants representing objects from the problem) with a 2-regulated domain model (each task decomposes either to an action or to two subtasks, similarly to Chomsky Normal Form). It demonstrates the benefits of grounding and it was shown to be faster on totally ordered domains than (Ondrčková et al. 2023) and (Höller et al. 2022). However, similarly to our approach, this approach does not support partially ordered domains.

The CYK-based approach inspired modification of existing parsing-based approach by pre-processing (Ondrčková et al. 2023). The authors showed that even greedy bottom-up parsing benefits from grounding with 2-regularisation. Bottom-up parsing was slower than the CYK-inspired algorithm on totally ordered domains; however, it supports also partially ordered domains.

Recently, the Earley parser has been suggested for hierarchical plan recognition (Pantůčková and Barták 2023), where it outperformed a bottom-up parsing approach and its performance was comparable to the compilation-based approach. This motivated us to apply the Earley parser also to hierarchical plan verification problem.

HTN plan verification by Earley parser

As an input of a plan verification problem, we expect a plan (a sequence of actions), a planning domain model (decomposition rules) and a description of a planning problem (initial state and possibly the goal task). We do not require the goal task to be given as part of the input. If the goal task is not known, we start with a dummy goal (root) task and for each abstract task from the domain model, we add to the domain model a new dummy rule decomposing the dummy goal into the abstract task.

The Earley parser is a parser for context-free grammars. A context-free grammar (CFG) is a grammar, where all rules can be written as $A \rightarrow \alpha$, where A is a non-terminal symbol and α is a string of terminal and non-terminal symbols. Omitting rule constraints in a totally ordered HTN planning

domain model results in a context-free grammar with rewriting rules of the form $T \rightarrow T_1, ..., T_n$, where T is an abstract task and $T_1, ..., T_n$ is a totally ordered sequence of abstract and primitive subtasks.

The Earley parser is based on three procedures: *predictor* decomposes abstract tasks, *scanner* reads actions from the input plan, and *completer* processes one subtask from a decomposition rule. The parser processes the input word from the left to the right. The computation is divided into iterations, each of which processes a prefix of the input word of an iteratively increasing length. In iteration i, the parser finds decompositions for the prefix of length i. We will not describe the Earley parser in detail as the complete algorithm can be found in (Earley 1970). Instead, we provide an explanation how the parser can be used for HTN plan verification.

Earley parser on grounded problems

On a grounded plan verification problem we can use the Earley parser algorithm directly – as described in (Earley 1970). As the input, the parser receives a sequence of actions (a plan) and a set of CFG rewriting rules. The parser decides whether the given word (the plan) belongs to the given CFG. The Earley parser finds all possible proofs – all abstract tasks which decompose into the given plan along with their decomposition trees. The decomposition trees can be ambiguous as there may be more ways how some abstract tasks can be decomposed. All possible decompositions of a task can be described by an AND/OR tree. OR-nodes correspond to abstract tasks, for each of which we must select one decomposition rule. AND-nodes represent decomposition rules, which decompose the compound task (the predecessor node) into the subtasks (the descendant nodes). Leaves correspond to primitive tasks (actions).

Each such task is a candidate goal. To decide whether a candidate goal can be decomposed into the given plan with respect to the decomposition rules of the original planning domain, we traverse its decomposition tree in a post-order DFS-like manner. For each abstract task, we successively visit all decomposition rules until we find a rule which can be used to decompose the task, or all rules are visited and rejected. A decomposition rule can be selected once all of its subtasks were visited, for each of which one decomposition rule has been selected. In order to select a decomposition rule, we have to verify that all rule constraints are satisfied. The complete algorithm verifying whether the rule constraints are satisfied can be found in (Barták, Maillard, and Cardoso 2020).

Earley parser on lifted problems

Our approach does not require a problem to be grounded. The Earley parser progresses from the top downwards, decomposing abstract tasks using all available rules until the actions are reached. If a problem is not grounded and the goal task is not given, the algorithm starts from the dummy root task G by using uninstantiated dummy rules $G \rightarrow \bullet T_i$, for each possible goal task T_i. When an action is reached, its attributes are propagated upwards to the rules higher in the decomposition tree. The partial instantiation of these rules will then be propagated downwards as the Earley parser progresses in order to reach the next action in the plan.

The *predictor* processes the state

$$T_i \rightarrow T_{i_1} ... T_{i_{j-1}} \bullet T_{i_j} ... T_{i_n},$$

which represents a rule decomposing the task T_i into the subtasks $T_{i_1}, ..., T_{i_n}$, where the tasks before $\bullet$ have already been processed and the next subtask to be processed is the abstract task T_{i_j}. The predictor creates a new state

$$T_j \rightarrow \bullet T_{j_1} ... T_{j_m},$$

for each decomposition rule of T_j and it propagates the attributes from tasks $T_i, T_{i_1}, ..., T_{i_{j-1}}$ to this new state $T_j \rightarrow \bullet T_{j_1} ... T_{j_m}$.

The *scanner* processes the state

$$T_i \rightarrow T_{i_1} ... T_{i_{j-1}} \bullet T_{i_j} T_{i_{j+1}} ... T_{i_n},$$

where T_{i_j} is a primitive task, whose instantiation is compatible with the action a, which is in the plan at the required position (based on the current iteration). The scanner creates a new state

$$T'_i \rightarrow T'_{i_1} ... T'_{i_{j-1}} a \bullet T'_{i_{j+1}} ... T'_{i_n},$$

where the relevant attributes are instantiated based on the attributes of a and the original instantiation of the state.

The *completer* processes the state

$$T_i \rightarrow T_{i_1} ... T_{i_n} \bullet,$$

using the completed decomposition to decompose the next task (the task right after $\bullet$) in all states, where the next task T_{j_k} is of the same type as T_i and with a compatible instantiation. For each such state

$$T_j \rightarrow T_{j_1} ... \bullet T_{j_k} ... T_{j_m},$$

the completer creates a new state

$$T'_j \rightarrow T'_{j_1} ... T'_{j_k} \bullet ... T'_{j_m},$$

with the instantiation resulting from merging the instantiations of $T_j, T_{j_1}, ..., T_{j_m}$ and T_i.

If any attribute of any decomposition rule remains uninstantiated while extracting a solution from the AND/OR graph, we try all possible instantiations until a valid grounded rule is found. If no such grounding exists, the rule is rejected.

Example

We provide now an example of the usage of the Earley parser for plan verification. Consider again the plan from Figure 1: *unstack(c,a), stack(b,a)* and consider the most difficult setting – lifted plan verification with unknown goal task.

The Earley parser successively generates states, each of which covers a continuous subsequence of the input plan. Apart from the rewriting rule with $\bullet$ separating decomposed and undecomposed subtasks, the Earley parser keeps for each state the index of the first and last action covered by the state. Therefore, a state is written in the following form:

$$[T_i \rightarrow T_{i_1} ... T_{i_{j-1}} \bullet T_{i_j} ... T_{i_n}; p; q],$$

where p is the index of the first and q is the index of the last action covered by this state, i.e., the tasks $T_{i_1}, ..., T_{i_{j-1}}$ decompose into the actions $a_p, ..., a_q$ from the plan.

As the goal task is not given as part of the input, the algorithm starts by generating a dummy starting state

$$[G \rightarrow \bullet T; 0; 0]$$

for each uninstantiated abstract task T from the domain, e.g.:

$$[G \rightarrow \bullet put\_on(?,?); 0; 0], [G \rightarrow \bullet clear(?); 0; 0].$$

Iteration 0: Predictor expands all starting states by applying all available decomposition rules to the first unprocessed subtask. One of the new states will be the following state:

$$[put\_on(?,?) \rightarrow \bullet clear(?) stack(?,?); 0; 0].$$

Then, it will expand also the newly generated states:

$$[clear(?) \rightarrow \bullet unstack(?,?); 0; 0].$$

As $unstack(?,?)$ is an action, the last state will be processed by scanner, which will read the first action and create a corresponding instantiated state:

$$[clear(a) \rightarrow unstack(c,a)\bullet; 1; 1],$$

which covers the action at the position 1. The new state will be processed in iteration 1 along with other generated states with the end index 1.

Iteration 1: The previous state will be processed by completer, which will use it to decompose the first subtask in the state $[put\_on(?,?) \rightarrow \bullet clear(?) stack(?,?); 0; 0]$ as $clear(?)$ is compatible with the instantiation $clear(a)$ and the end index (0) precedes the index 1. As a result, it will create the following state:

$$[put\_on(?,a) \rightarrow clear(a) \bullet stack(?,a); 0; 1].$$

As $stack(?,a)$ is an action, this state will be processed by scanner, which will read the second action from the plan, thus creating the following state:

$$[put\_on(b,a) \rightarrow clear(a) stack(b,a)\bullet; 0; 2].$$

Iteration 2: The states with the end index equal to 2 will be processed, along with the previous state, which will be used by completer to complete one of the dummy starting states, creating the corresponding state:

$$[G \rightarrow put\_on(b,a)\bullet; 0; 2].$$

As the leaves of the decomposition AND/OR tree of this state contain all actions, this state represents a candidate goal task.

As we remember which rule was used to decompose each subtask of each state (e.g., $clear(a) \rightarrow unstack(c,a)$ was used to decompose the first subtask of $put\_on(b,a) \rightarrow clear(a) stack(b,a)$), we may now attempt to extract a solution from the AND/OR tree of the candidate goal. We traverse the tree using the post-order depth-first search, firstly visiting the root task $put\_on(b,a)$. The decomposition rule that decomposes this task into $clear(a)$ and $stack(b,a)$ can be used only if all subtasks can be decomposed. Hence, we have to firstly visit the task $clear(a)$. We will now verify that $clear(a)$ can be decomposed into $unstack(c,a)$, i.e., that no constraint of the decomposition rule is violated and all preconditions of the action $unstack(c,a)$ are satisfied in the initial state (since it is the first action in the plan), e.g., that the block c is on the block a and no block is on the block c. If the decomposition is verified, we return back to the task $put\_on(b,a)$ and verify that it can be decomposed into the abstract task $clear(a)$ and the action $stack(b,a)$. For example, we have to verify that the preconditions of the action $stack(b,a)$ are satisfied in the state after the first action $unstack(c,a)$ is executed, i.e., no block is on the block b and no block is on the block a. If this decomposition is valid, the root task is the goal that decomposes into the given plan; therefore, the plan is valid. If no such goal can be found, the plan is invalid.

The grounded parsing progresses in a similar manner. Instead of using uninstantiated or instantiated rules, we select rules with the required variables. If the goal is given as part of the input, we create only one dummy starting state $[G \rightarrow \bullet T; 0; 0]$ for the given goal task T.

Empirical evaluation

We have compared the performance of the Earley verifier with three state of the art hierarchical plan verifiers: the compilation-based approach (Höller et al. 2022), the bottom-up parsing-based approach (Ondrčková et al. 2023), and the CYK-based approach (Lin et al. 2023). We have compared the performance of the Earley parser on grounded and lifted domains. All three existing approaches and the grounded Earley parser use the grounder from (Behnke et al. 2020).

As benchmarks, we have used three totally ordered domains from IPC 2020: Satellite, Transport and Blocksworld. As valid plans, we have used the valid plans from IPC 2020. We have used 45 valid plans of length 5 - 28 from the domain Satellite, 50 plans of length 8 - 225 from the domain Transport and 28 plans of length 22 - 357 from the domain Blocksworld and the same number of invalid plans that were created by deleting trailing actions from valid plans. The benchmarks used for experiments will be accessible online after the paper is accepted. All experiments were performed on a computer with the Intel Core i7-8550U CPU @ 1.80GHz processor and 16 GB of RAM. Maximum allowed runtime was set to five minutes for one problem.

Figures 2, 3, 4, 5, 6 and 7 show the results. We have compared how the number of problems solved by each verifier increases with runtime separately for valid and invalid plans from each domain. In each figure, *Earley* denotes the lifted Earley parser, *CYK* denotes the CYK-based approach (Lin et al. 2023), *compilation* denotes the compilation-based approach (Höller et al. 2022), *BFS* denotes the bottom-up parsing-based approach (Ondrčková et al. 2023), and *grounded Earley* denotes the grounded Earley parser.

Generally, the grounded Earley parser was usually slightly slower than both bottom-up parsing-based approaches; however, the differences of runtimes were usually counted only in hundreds of milliseconds. The compilation-based approach was faster on invalid plans, but slower than all three

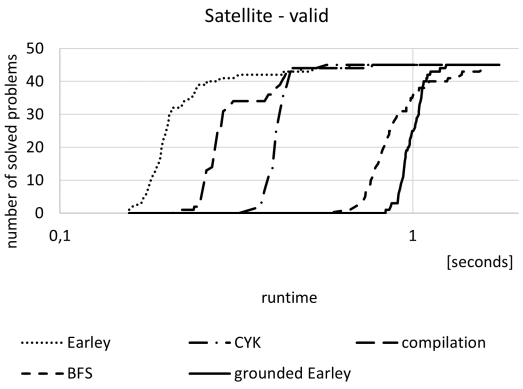

Figure 2: The number of problems with valid plans solved within given time (log x-axis) in the domain Satellite.

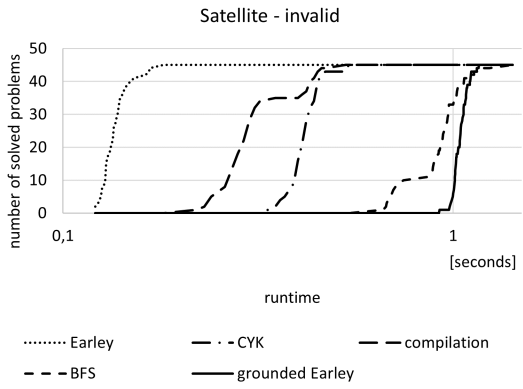

Figure 5: The number of problems with invalid plans solved within given time (log x-axis) in the domain Satellite.

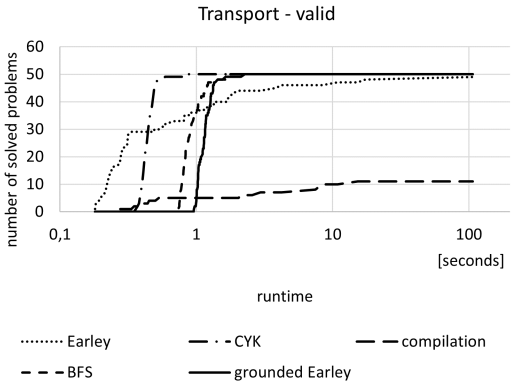

Figure 3: The number of problems with valid plans solved within given time (log x-axis) in the domain Transport.

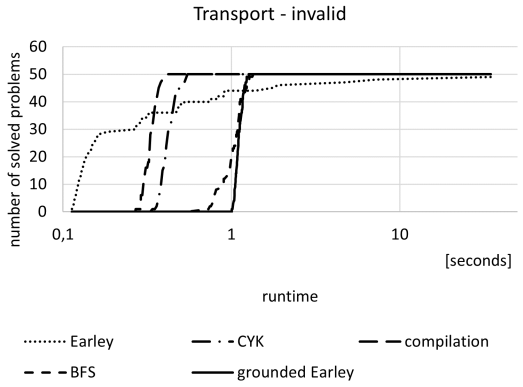

Figure 6: The number of problems with invalid plans solved within given time (log x-axis) in the domain Transport.

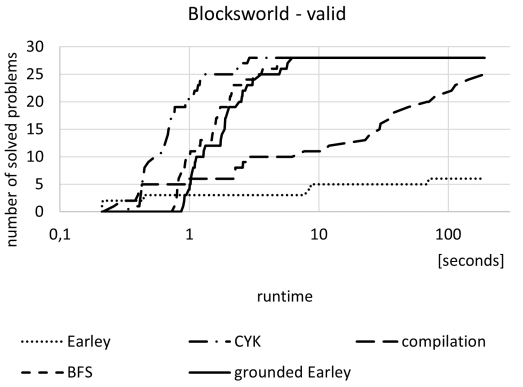

Figure 4: The number of problems with valid plans solved within given time (log x-axis) in the domain Blocksworld.

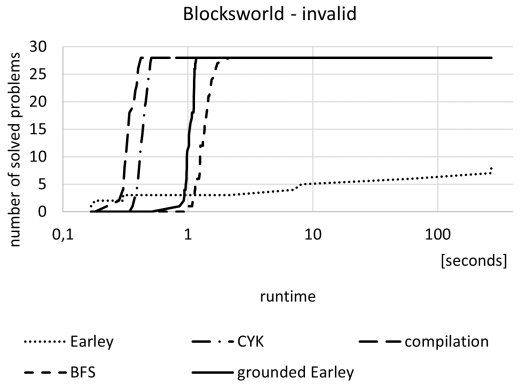

Figure 7: The number of problems with invalid plans solved within given time (log x-axis) in the domain Blocksworld.

grounded parsing-based approaches on longer valid plans from the domains Transport and Blocksworld.

The lifted Earley parser was significantly slower than all of the other verifiers on longer both valid and invalid plans (see Figures 3, 4, 6 and 7). In the domain Blocksworld (Figures 4 and 7), the lifted Earley parser even failed to solve most of the problems within the maximum allowed runtime.

However, the lifted Earley parser was faster on shorter

plans. In the domain Satellite, it solved most of the problems faster than all other verifiers (see Figures 2 and 5); the difference is bigger for invalid plans (Figure 5). In this domain, consisting of the shortest plans, the grounding results in an unnecessary overhead. A similar observation can be made also in the domain Transport (Figures 3 and 6). For both valid and invalid plans, the lifted verifier solved more than half of the problems faster than the other verifiers, but when it was faced with more difficult problems with longer plans, its performance decreased. On longer plans from the domain Transport and on the domain Blocksworld, consisting of the most difficult problems, grounding seems to prune the space of intermediate solutions, which results in a significant increase of performance of plan verification.

Though the Earley parser did not show a better performance than the existing parsing-based approaches to plan verification, the fact that its performance was comparable opens directions for future research. The Earley parser is based on top-down parsing, while the other approaches are based on bottom-up parsing. As the Earley parser parses the input from the top downwards, progressing from abstract tasks to actions via decomposition rules in a similar manner as HTN planning, a question arises whether the performance of verification could be improved by HTN planning heuristics, which can hardly be implemented in bottom-up parsing. Implementation of planning heuristics into parsing-based plan verification could be interesting especially since parsing-based approaches seem to perform better than compilation when applied to plan verification.

Conclusion

We have presented a top-down parsing approach to hierarchical plan verification based on the Earley parser – a parser originally proposed for context-free grammars. We have evaluated the effect of grounding on the performance of our approach and we have shown that grounding significantly improves its performance on more difficult problems. We have shown that the performance of our top-down parsing approach is comparable to the performance of the existing bottom-up parsing approaches. This result suggests to explore the possible implementation of hierarchical planning approaches into parsing-based hierarchical plan verification. Since top-down parsing resembles hierarchical planning, planning heuristics can be implemented more naturally than in bottom-up parsing.

Acknowledgments

Research is supported by the Charles University, project GA UK number 156121, by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215 and by SVV project number 260 698.

References

Barták, R.; Ondrčková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A novel parsing-based approach for verification of hierarchical plans. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, 118–125. IEEE.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of hierarchical plans via parsing of attribute grammars. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS*, 11–19. AAAI Press.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2020. Parsing-based approaches for verification and recognition of hierarchical plans. In *The AAAI 2020 Workshop on Plan, Activity, and Intent Recognition*.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of htn planning problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9775–9784.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This is a solution!(... but is it though?)-verifying solutions of hierarchical planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, 20–28.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 13(2):94–102.

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

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling htn plan verification problems into htn planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 145–150.

Howey, R., and Long, D. 2003. Val's progress: The automatic validation tool for pddl2.1 used in the international planning competition. In *Proceedings of the ICAPS 2003 workshop on "The Competition: Impact, Organization, Evaluation, Benchmarks"*, 28–37.

Lin, S.; Behnke, G.; Ondrčková, S.; Barták, R.; and Bercher, P. 2023. On total-order htn plan verification with method preconditions–an extension of the cyk parsing algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 12041–12048.

Ondrčková, S.; Barták, R.; Bercher, P.; and Behnke, G. 2022. On heuristics for parsing-based verification of hierarchical plans with a goal task. In Barták, R.; Keshtkar, F.; and Franklin, M., eds., *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022*.

Ondrčková, S.; Barták, R.; Bercher, P.; and Behnke, G. 2023. Lessons learned from the cyk algorithm for parsing-based verification of hierarchical plans. In *The International FLAIRS Conference Proceedings*, volume 36.

Pantůčková, K., and Barták, R. 2023. Using earley parser for recognizing totally ordered hierarchical plans. In *Proceedings of 26th European Conference on Artificial Intelligence (ECAI) 2023*. IOS Press. 1819–1826.