Compiling HTN Plan Verification Problems into HTN Planning Problems

Daniel Höller[1], Julia Wichlacz[1], Pascal Bercher[2], and Gregor Behnke[3,4]

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]The Australian National University, Canberra, Australia
[3]University of Freiburg, Freiburg, Germany
[4]University of Amsterdam, ILLC, The Netherlands

hoeller@cs.uni-saarland.de, wichlacz@cs.uni-saarland.de, pascal.bercher@anu.edu.au, g.behnke@uva.nl

Motivation

- In HTN planning, the decomposition steps applied to find a solution are usually not part of the solution

→ The solution is a sequence (partially ordered set) of actions

- This makes verifying whether a solution returned by a planner is correct a hard task (up to NP-hard)
- Given certain mappings and the steps, it becomes polynomial

→ E.g. for the IPC, systems needed to provide them

- However, they are often not needed and tracking them is sometimes (technically) complicated
- Sometimes they are even not available at all (e.g. after post-processing, or in compilation-based planning systems)

Plan Verification – Related Work

Approaches from the literature:

- Translation to propositional logic (Behnke et al. 2017)
- Based on parsing (see e.g. Barták et al. 2018)

Plan Verification – Approach

We present a compilation to HTN planning

- We compile the plan verification problem into an HTN planning problem that has a solution if and only if the solution is correct
- We use HTN planning systems to solve the compiled problem
- The used planners return the decomposition steps

→ Witness for correctness

Plan Verification

Given in the problem definition:

- HTN planning problem p
- sequence of actions $(a_1 a_2 \dots a_n)$

Task: Decide whether there is a solution $(T_S, \prec_S, \alpha_S)$

- for an ordering $(i_1 i_2 \dots i_n)$ of task identifiers
- it holds that $(\alpha_S(i_1) \dots \alpha_S(i_n)) = (a_1 \dots a_n)$
- Based on the original planning problem...
- ...we generate a problem in which exactly the given sequence is executable

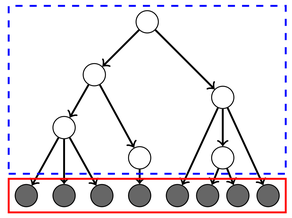

Figure 1: *Plan verification*

Plan Verification – Compilation

Step 1:

- Copy the actions in the solution to verify: $a \to a', a''$
- a' gets new preconditions and effects, it can only be placed at the respective position in the solution
- All actions have to be in the plan

Step 2 is illustrated in Figure 3.

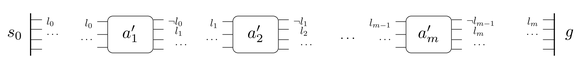

Figure 2: *Prefix Compilation*

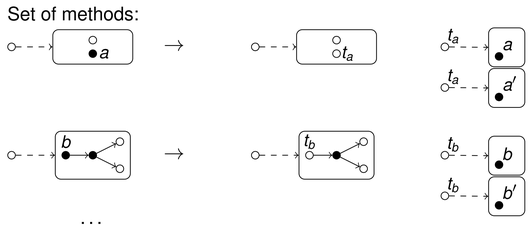

Figure 3: *Example for method transformation (plan to verify: $abab$)*

Evaluation – Systems

- We use two off-the-shelf planning systems from the PANDA framework to solve the resulting problems
- They return a witness for the decomposition process
- This allows to check of correctness in P
- We compare our approach against SAT-based and parsing-based systems from the literature

Evaluation – Benchmark Set

- We created a benchmark set based on the 2020 IPC models
- We included plans generated by the participants of the IPC and from systems from the PANDA framework
- To also include non-solutions, we included incorrect plans returned by non-final versions of the IPC participants (before the debugging process)
- Problem for verification: method preconditions

Method Preconditions

- State-based preconditions of methods are very common in HTN planning
- To make their support as simple as possible for planners, the HDDL standard defines its semantics via a compilation
- However, this leads to problems for verifiers (see Figure 4)
- For totally ordered problems, this is not a problem

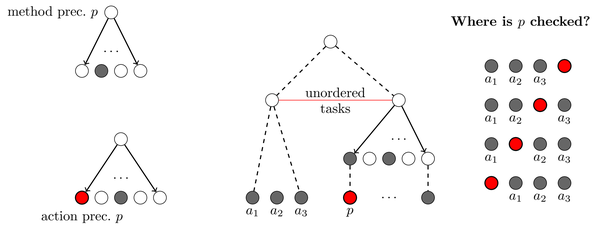

Figure 4: *Method preconditions*

Method Preconditions

- SAT-based and parsing-based approaches do not (fully) support method preconditions
- For our approach, they are not a problem: model parts concerned with the method preconditions just need to be ignored
- However, this forms a problem for the evaluation
- We created a second benchmark set by removing the method preconditions from all models
- Since this might change the characteristics of the problems, we ran all verification systems on both sets

Results

- Compilation with progression search-based planner performs best on the original benchmark set
- SAT-based planner has problems showing unsolvability
- Coverage of all systems drops on the models where method preconditions have been removed

		Inst.	Compilation		Parsing	SAT
			Progression	SAT		
TO	Valid	10961	**10881 (99.27)**	9757 (89.02)	9158 (83.55)	not supported
	Invalid	1406	**1364 (97.01)**	727 (51.71)	1301 (92.53)	not supported
PO	Valid	1211	1088 (89.84)	**1198 (98.93)**	not supported	not supported
	Invalid	138	**129 (93.48)**	64 (46.38)	not supported	not supported
TO No M.P.	Valid	11304	**9679 (85.62)**	8986 (79.49)	7889 (69.79)	1036 (9.16)
	Invalid	1063	898 (84.48)	406 (38.19)	**915 (86.08)**	684 (64.35)
PO No M.P.	Valid	1243	1103 (88.74)	**1212 (97.51)**	973 (78.28)	897 (72.16)
	Invalid	106	98 (92.45)	57 (53.77)	**106 (100.00)**	103 (97.17)

Figure 5: *Coverage results*

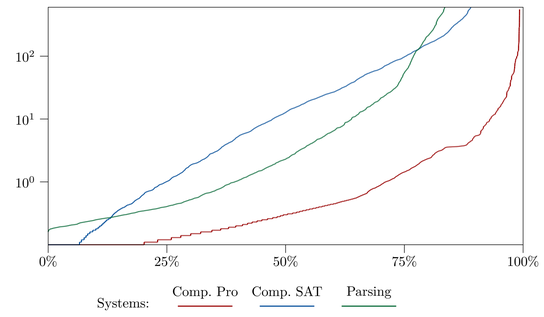

Figure 6: *Runtime against solved instances (setting: valid solutions, TO with method preconditions)*

Conclusion

- We have introduced a compilation-based approach to HTN plan verification
- We used off-the-shelf planners to solve the problems resulting from the compilation
- Our planners return the information enabling verification in P
- We introduced a novel benchmark set for plan verification based on the models from the 2020 IPC
- Our system is the only one supporting all language features used in the IPC
- Our system outperforms the systems from the literature

UNIVERSITÄT DES SAARLANDES