Lifted Logic for Task Networks: TOHTN Planner Lilotane Entering IPC 2020

Dominik Schreiber
Karlsruhe Institute of Technology
dominik.schreiber@kit.edu

Abstract

We present our contribution to the International Planning Competition (IPC) 2020. Our planner *Lilotane* builds upon ideas established by Tree-REX and encodes a totally ordered hierarchical task network (TOHTN) planning problem into incremental formulae of propositional logic (SAT). *Lilotane* features lazy instantiation and encodes reductions and actions without the need for full grounding, hence accelerating the planning process significantly.

Overview

In this announcement we briefly describe *Lilotane* (ˈlɪ·lo·teɪn, *Lifted Logic for Task Networks*), the first Satisfiability (SAT) based planner for totally ordered Hierarchical Task Network (TOHTN) problems that operates on a partially lifted planning problem. The general planning procedure of our planner is similar to the planning pipeline known from its predecessor Tree-REX (Schreiber et al. 2019) as well as from totSAT (Behnke, Höller, and Biundo 2018):

1. The formal description of a planning problem is parsed and preprocessed in some way.
2. Propositional logic clauses describing the problem's upmost yet unencoded *hierarchical layer* are added, a fully expanded task network is assumed, and a SAT solver is run on the resulting formula.
3. If the solver finds a model, a plan is decoded from the satisfying assignment to the Boolean variables and returned. Otherwise, go to 2.

The main difference between previous approaches and *Lilotane* is that the latter does not perform a complete grounding of the problem in step 1; instead we perform lazy instantiation of operators and methods in step 2, just in time for when they are needed. Secondly, we avoid to instantiate all parameters of an action or a reduction occurring at some place of the hierarchy, instead introducing *pseudo-constants* whose semantics we define directly in propositional logic. We describe these techniques in the next two sections.

Our planner is written from scratch in C++. In the competition version we use SAT solver Glucose (Audemard and Simon 2009). We build upon *pandaPIparser* (Behnke et al. 2020) for parsing planning problems specified in HDDL and for preprocessing the problem's lifted representation. *Lilotane* is free software licensed under the GNU General Public License (GPL) v3.0[1].

Instantiation

When receiving a parsed and preprocessed problem description from *pandaPIparser*, we instantiate the problem layer by layer, just in time for each solving attempt. For the exact definition of these layers, we refer to (Schreiber et al. 2019) where the same layout of layers was used but its construction was based on a problem's full grounding.

We instantiate each layer in chronological order ("from left to right"), remembering the position $p \in \mathbb{N}_0$ where each instantiated reduction and action was found. While instantiating a layer we maintain sets of positive and negative facts which may possibly occur, beginning with the initial state and adding any direct or indirect effects of an instantiated operation. Using preconditions and constraints of method and operator definitions, we can use these fact collections to prune all operations with impossible preconditions. We determine the possible effects of a given operation using a conservative (false positive) lookup table which we only need to compute once per operator and per method. In addition, we logically infer new preconditions for a method by recursively aggregating the preconditions and effects of its possible children: This helps us to profit from the described pruning methods even on domains which do not natively feature any method preconditions.

Furthermore, we significantly reduce the number of instantiated actions and reductions by introducing pseudo-constants. Consider an example task `(navigate ?rover ?from ?to)` which, according to its parent task `(investigate A)`, evaluates to `(navigate ?rover ?from A)`. Performing a conventional instantiation we receive tasks `(navigate R1 B A)`, `(navigate R1`

[1] Additional legal constraints may apply depending on the licensing of the particular SAT solver *Lilotane* is linked with.

`C A)`, `(navigate R1 D A)`, `(navigate R2 B A)` and so on. Our algorithm avoids this blowup, for instance by instantiating only one task: `(navigate !R !FROM A)`. Thereby, `!R` and `!FROM` are new symbols which did not occur in the problem before and which we call *pseudo-constants*. Logically this task is still lifted as it contains arguments that are not committed to a particular constant in the problem yet. With our novel SAT encoding we can let the solver decide which constants to assign. Note that our encoding and solving approach can handle any amount of "ground-ness" produced during instantiation, ranging from fully ground objects to fully lifted representations. In the competition version, our instantiation algorithm aggressively introduces pseudo-constants, only fully instantiating free arguments in some edge cases.

Encoding

The general structure of our propositional logic encoding is taken from (Schreiber et al. 2019). The main difference is that we now must deal with actions, reductions, and facts containing pseudo-constants. Here we provide some central clause definitions but refer to a future full paper for the complete specification of the encoding.

As in previous work we use one Boolean variable for each occurring reduction, action, and fact per position per layer of the problem. These variables are assigned regardless of whether the object contains pseudo-constants or not. Also, we have one variable $\mathit{primitive}(l, i)$ representing whether position i at layer l features a primitive operation, i.e., an action and not a reduction.

In addition, we introduce global variables $[!p/c]$ that correspond to substituting some pseudo-constant $!p$ with an actual constant c. For each pseudo-constant $!p$ we add clauses

$$\bigvee_{c \in dom(!p)} [!p/c] \quad \wedge \quad \bigwedge_{c_1, c_2 \in dom(!p)} \neg[!p/c_1] \vee \neg[!p/c_2],$$

i.e., exactly one of the possible substitutions of $!p$ with a constant from its possible domain, $dom(!p)$, must hold.

Next, we define the semantics of facts containing pseudo-constants, which we call pseudo-facts. Let $!f$ be a pseudo-fact and for each of its pseudo-constants $!p$ let $c_{!p}$ be one of the possible constants to be substituted.

$$\Big(\bigwedge_{!p \in !f} [!p/c_{!p}]\Big) \Rightarrow \big(\mathit{holds}(!f, l, i) \Leftrightarrow \mathit{holds}(!f_{[!p/c_{!p}]}, l, i)\big)$$

In words, we enforce a pseudo-fact to be equivalent to the ground fact it corresponds to when performing particular substitutions. This rule does imply that we need to fully instantiate all potentially occurring facts at the respective position; yet, there are commonly much fewer ground facts in a problem than there are actions or reductions.

Frame axioms are encoded only for fully ground facts, as the meaning of pseudo-facts is well-defined by the previous sets of clauses. We add clauses as follows:

(i) If a fact f changes its value, then either some anonymous reduction is responsible for the change (denoted with $\neg\mathit{primitive}(l, i)$), or some action *directly* supports this fact change, or some pseudo-action *indirectly* supports the fact change. (ii) If fact f changes its value and some pseudo-action $!a$ is applied, then some set of substitutions must be active which unify some effect $!f$ of $!a$ with f.

Note that for (ii), in the general case a transformation of a Disjunctive Normal Form (DNF) into Conjunctive Normal Form (CNF) is required when $!a$ features many different pseudo-facts as effects which can be unified to f. This may become an issue if an HTN domain features a large number of effects of the same predicate in a single action.

We add additional kinds of clauses which are constraining the set of possible substitution combinations (due to preconditions which must hold), retroactively restricting the domain of a pseudo-constant to incorporate argument type restrictions of operations, and switching off certain negative action effects in the case where the same fact also occurs as a positive effect in the action.

Closing Remarks

We consider our new encoding to be structurally more complex than that of Tree-REX but observed empirically that our approach not only significantly cuts the time spent for instantiation but also leads to much smaller formulae, oftentimes by orders of magnitude. We argue that the approach to outsource logical aspects of grounding to the SAT encoding equips the solver with structural insights which an encoding generated from ground representations cannot have.

We close with mentioning that *Lilotane* is designed for agile planning and hence does not optimize plan length (i.e., the number of actions). Yet, *Lilotane* is guaranteed to find the plan of lowest possible (hierarchical) *depth*, and we also intend to integrate a plan length optimization similar to the anytime SAT-based plan optimization from Tree-REX.

Acknowledgments

The author would like to thank Marvin Williams for his persevering exploration of lifted SAT encodings for classical planning (Williams 2020), thus providing inspirations to pursue a lifted encoding for HTN planning as well.

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Twenty-first International Joint Conference on Artificial Intelligence*.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of HTN planning problems. In *AAAI*, 9775–9784.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT-totally-ordered hierarchical planning through SAT. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Schreiber, D.; Pellier, D.; Fiorino, H.; et al. 2019. Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 382–390.

Williams, M. 2020. Partially instantiated representations for automated planning. Master's thesis, Karlsruhe Institute of Technology.