Lifted Logic for Task Networks: TOHTN Planner Lilotane in the 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*, however, instantiates reductions and actions lazily and minimalistically without the need for full grounding, hence accelerating the planning process significantly. We discuss the results of the IPC and conclude that *Lilotane*, scoring second in the Total Order track, is an overall competitive system, what demonstrates the viability of our approach and its significance for future research.

Overview

In this report we present *Lilotane* ('lı·lo·teın, *Lifted Logic for Task Networks*), the first Satisfiability (SAT) based planner for Totally Ordered Hierarchical Task Network (TOHTN) problems that operates on a lifted planning problem. The design of Lilotane is heavily motivated by the observation that grounding an HTN planning problem (Ramoul et al. 2017; Behnke et al. 2020) induces an unavoidable worst-case combinatorial blowup with respect to the input size, and that this blowup can hinder SAT-based HTN planners to scale to larger problems even if they are logically of simple nature. *Lilotane*, by contrast, fully circumvents the stage of grounding and instead encodes a lifted problem representation into propositional logic.

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 $\Pi = (D, s_I, T)$, where D is an HTN planning domain, s_I is an initial state, and T is a sequence of initial tasks, is parsed and preprocessed in some way.
2. Propositional logic clauses describing the problem's upmost yet unencoded *hierarchical layer* L_l 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.

Algorithm 1: Lilotane Procedure (simplified)

Input: $\Pi = (D, s_I, T)$
Result: Plan π
Preprocess Π; `// parsing, simplification`
$H := \langle\rangle$;
$L_0 := \langle$ CreateInitialPosition(T, s_I) $\rangle$;
Encode(L_0); `// encode first layer`
$H := H \circ \langle L_0 \rangle$;
for $l = 0, 1, \ldots$ **do**
  `// instantiate new layer`
  $L_{l+1} := \langle\rangle$;
  $S := (s_I, \emptyset)$; `// reachable facts`
  $x' := 0$;
  for $x = 0, \ldots, |L_l| - 1$ **do**
    `// generate child positions of` $P_{l,x}$
    $e_{l,x} := \max\{1, \max\{|\mathit{subtasks}(r)| \mid r \in P_{l,x}\}\}$;
    for $z = 0, \ldots, e_{l,x} - 1$ **do**
      $P_{l+1,x'} :=$ Instantiate$(P_{l,x}, z, S)$;
      $L_{l+1} := L_{l+1} \circ \langle P_{l+1,x'} \rangle$;
      $S := S \cup \mathit{possibleFactChanges}(P_{l+1,x'})$;
      $x' := x' + 1$;
    end
  end
  `// encode new layer`
  Encode(L_{l+1});
  `// finalize layer, attempt to solve`
  $H := H \circ L_{l+1}$;
  result := Solve(H);
  if *result is SAT* **then**
    return Decode(H, result);
  end
end

The main difference between previous approaches and *Lilotane* is that the latter avoids the 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. We avoid to instantiate all free arguments of an action or a reduction occurring at some place of the hierar-

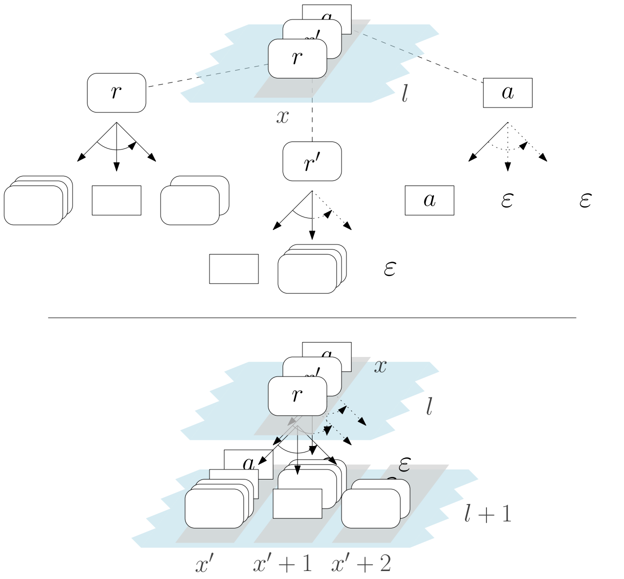

Figure 1: Sketch of *Lilotane*'s instantiation procedure. Above: Position x at layer l contains three operations (reductions r and r', and action a) with different possible children. Below: New positions $x', \ldots, x'+2$ are appended to layer $l+1$ by aggregating the respective possible children.

chy. Instead we introduce *pseudo-constants* whose semantics we define directly in propositional logic. In this report, we briefly elaborate on these techniques in the upcoming sections, provide some technical insights, discuss the performance of *Lilotane* within the IPC, and provide a brief conclusion and outlook. For a more detailed presentation and discussion of the work at hand we refer to Schreiber (2021).

Instantiation

The base algorithm of our approach is illustrated in Alg. 1. After receiving the lifted problem description from *pandaPIparser*, we instantiate the problem's hierarchy from top to bottom, i.e., we begin with an initial layer following from the problem description (line 3) and then construct layer $l+1$ on the basis of layer l. Each operation (i.e., action or reduction) at some position of layer l can induce one or several new positions at layer $l+1$. Each such new position may again feature a variety of different operations, as illustrated in Fig. 1. Only one such operation at each position will be chosen by a SAT solver for the final plan. This approach is based on *Tree-REX* (Schreiber et al. 2019) where the same layout of layers was used but its construction was based on a problem's full grounding. By contrast, we instantiate operations just when needed to achieve some subtask, and we preserve free arguments of methods instead of instantiating them with all possible combinations of constants.

As we instantiate each layer in chronological order ("from left to right"), we can maintain sets S of positive and negative facts which may possibly occur, beginning with the initial state (line 8) and adding any direct or indirect effects of inserted operations (line 15). We can use these fact collections to discard any operations with a precondition that turns out to be impossible to achieve in line 13. In line 15 we determine the possible effects of a given operation using a conservative overestimation which we compute by a traversal of the (lifted) recursive children of a 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.

By allowing for free arguments to remain in an operation, we significantly reduce the number of instantiated actions and reductions. 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 C A)`, `(navigate R1 D A)`, `(navigate R2 B A)` and so on. Our algorithm avoids this blowup by instantiating only one task: `(navigate` α β `A)`. Thereby, α and β are new symbols which did not occur in the problem before and which we call *pseudo-constants*. With our novel SAT encoding we can let the solver decide which particular constant to substitute each pseudo-constant with. Our instantiation algorithm introduces a pseudo-constant whenever the valid domain of a free variable is larger than one, i.e., whenever there is a nontrivial choice to make regarding the substitution.

We introduced several further techniques to increase performance, such as (i) the sharing of pseudo-constants among multiple operations and the notion of an operation *dominating* another operation if it represents a superset of ground operations; (ii) the retroactive pruning of any subtree of operations which turned out to be impossible to achieve; (iii) the transformation of certain reductions into equivalent actions; and more.

Encoding

The general structure of our propositional logic encoding is an adaptation of the *Tree-REX* encoding (Schreiber et al. 2019). The main difference is that we now must deal with actions, reductions, and facts containing pseudo-constants. We now provide some central, slightly simplified clause definitions for illustration purposes and refer to Schreiber (2021) for the complete specification.

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 *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 $[\kappa/c]$ that correspond to substituting some pseudo-constant κ with an actual constant c. For each pseudo-constant κ we add clauses

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

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

Next, we define the semantics of facts containing pseudo-constants, which we call pseudo-facts. Let f_p be a pseudo-fact and for each of its pseudo-constants κ let c_κ be one of the possible constants to be substituted such that substituting each κ with c_κ leads to ground fact f.

$$\Big(\bigwedge_{\kappa \in f_p} [\kappa/c_\kappa]\Big) \Rightarrow \big(holds(f_p, l, i) \Leftrightarrow holds(f, l, i)\big)$$

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, we claim that there are commonly much fewer ground facts than there are actions or reductions.

Frame axioms are encoded only for 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 the position is non-primitive, 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 action a from the indirect support is applied, then some set of substitutions must be active which unify some effect f_p 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. We use a simple compilation which builds a tree of literals and then obtains CNF clauses by traversing it.

Compared to a SAT encoding based on a ground representation, there are some subtle new edge cases to consider. For instance, we need to add further clauses which constrain the sets of possible substitution combinations (due to invariant preconditions which we do not encode directly), retroactively restrict the domain of a pseudo-constant to incorporate argument type restrictions of a child operation, and conditionally disable certain negative action effects if an equivalent fact also occurs as a positive effect in the action.

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.

Technical Remarks

Our planner is written from scratch in C++ (i.e., we did not reuse any code from previous planners). In the competition version we use SAT solver Glucose (Audemard and Simon 2009) with kind permission of the authors: Empirically we found this solver to work best on the class of formulae generated by our approach. We build upon *pandaPIparser* (Behnke et al. 2020) for parsing planning problems specified in HDDL and for performing light preprocessing tasks on the problem's lifted representation. *Lilotane* is free software licensed under the GNU General Public License (GPL) v3.0; additional legal constraints may apply depending on the licensing of the particular SAT solver

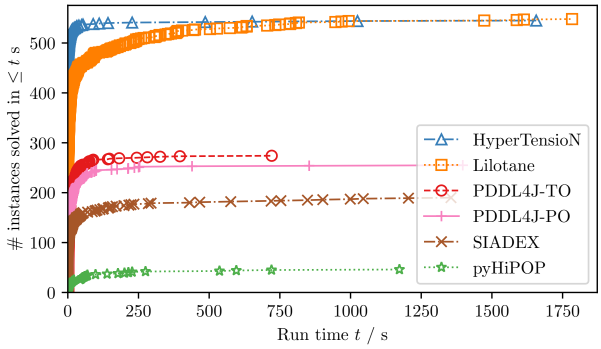

Figure 2: Run times overview of the IPC Total Order track. Each point (t, y) of participant p corresponds to an instance solved at least once by p in t seconds on average.

Lilotane is compiled with. Our software is available at `github.com/domschrei/lilotane`.

Post-IPC Discussion

We now discuss the results of the International Planning Competition (IPC) 2020.

A large set of diverse benchmark problems from various authors was gathered for the IPC, what will certainly facilitate thorough evaluations of TOHTN planners in the future. Compared to most previous evaluations in TOHTN planning (Schreiber et al. 2019; Behnke, Höller, and Biundo 2018) we observed that the peak difficulty of problems has been increased substantially: Oftentimes a domain known from previous evaluations was extended by ten more instances, each of which larger than any previous instance. This means that a planner reaching a near-perfect score on some domain is generally a much stronger result than before.

Lilotane scored the second place in the Total Order track of the IPC 2020. It found a plan for 548 out of 892 instances in at least one out of ten repetitions and reached an IPC score of 11.6. *Lilotane* was outperformed by progression search planner *HyperTensioN* which reached a considerably better score of 13.51 and found a plan for 545 instances in at least one repetition. *HyperTensioN* solved 84% of its instances in less than a second. *Lilotane* only solved 41% of its instances in under a second and solved 84% in under one minute. Overall we observed that while the IPC score benefits the overall much faster execution times of *HyperTensioN*, *Lilotane* performed similarly to *HyperTensioN* in terms of robustness and, unlike *HyperTensioN*, was able to solve some instance(s) on every single domain.

All further competitors scored significantly lower. In particular, *Lilotane* outperformed the only ground approach participating, *PDDL4J*, on all but four domains. *HyperTensioN* scored best on 15/24 domains and *Lilotane* scored best on 8/24 domains; only a single domain (Entertainment) was neither won by *HyperTensioN* nor by *Lilotane*.

Lilotane's worst performances are on the domains Blocksworld-HPDDL, Minecraft(-Player), and Multiarm-

Blocksworld. We noticed that each of these domains leads to deep and large hierarchical task networks which favor greedy progression search planners over planners such as *Lilotane* which are required to instantiate the entire hierarchy with all possible alternatives up to the layer where a plan can be found. Furthermore, compiled universal quantifications in Blocksworld-HPDDL and Multiarm-Blocksworld lead to many preconditions per operator which are comparably costly for our encoding.

By contrast, our planner excelled on domains such as Monroe (complex goal and task recognition problems on top of a disaster management domain, see Höller et al. 2018) and Woodworking. The latter domain encompasses large manifacturing and processing tasks and notably features a high number of arguments per operator and method. As our approach keeps free arguments lifted, it can handle this domain very well. We are also pleased to observe that *Lilotane* scored well on the Childsnack domain: This domain is a textbook example for a logically trivial domain which leads to huge ground representations. Hence, prior SAT-based approaches have considerable problems with this domain while our approach solves even large problems with relative ease.

Although the IPC was an agile competition where only run times were of interest, we also want to shed light on the length of the plans found by the best competitors (with respect to the number of actions in a plan). We found that *Lilotane* produced considerably shorter plans than the winner: We filtered out all 439 instances for which both *Lilotane* and *HyperTensioN* found a plan on some runs and then averaged the found plan length over all successful runs for each instance. On 264 instances *Lilotane* found shorter plans on average, on 77 instances the found plans are of equal average length and on 98 instances *HyperTensioN* found shorter plans on average. Summed up over all these instances, the number of actions reported by *HyperTensioN* corresponds to 229% of the number of actions reported by *Lilotane*. This significant difference in plan quality can be explained by the careful iterative deepening procedure of *Lilotane*: Any found plan length is bounded by the size of the layer where it was found, and *Lilotane* finds a plan on the very first layer where any plan *can* be found.

Conclusion and Outlook

We presented our submission to the IPC 2020 named *Lilotane* which is the first lifted SAT-based HTN planning system. *Lilotane* showed promising performance and convinced on a large and diverse set of benchmarks with respect to its robustness and the high-quality plans it finds. As such, the performance of *Lilotane* in the IPC 2020 demonstrates that SAT-based HTN planning without grounding is not only viable, but in fact a highly appealing approach if done carefully. We expect these results to open up new perspectives for SAT-based planning in related problem classes. We refer to a separate article (Schreiber 2021) which discusses the research at hand in more detail, provides proofs of correctness, and describes further improvements of *Lilotane* integrated after the planner submission deadline of the IPC.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

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

Furthermore, the author thanks the IPC organizers for their diligent work on this important competitive event and specifically Gregor Behnke for fruitful discussions regarding the results of the IPC 2020 and for providing an HDDL parser the author made thankful use of.

Last but not least, many thanks to Damien Pellier, Humbert Fiorino, and Tomáš Balyo who introduced the author to the exciting topic of SAT-based planning and HTN planning.

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Twenty-first International Joint Conference on Artificial Intelligence*, 399–404.

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. 2018. totSAT – totally-ordered hierarchical planning through SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 6110–6118.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018. Plan and goal recognition as HTN planning. In *30th International Conference on Tools with Artificial Intelligence*, 466–473. IEEE.

Ramoul, A.; Pellier, D.; Fiorino, H.; and Pesty, S. 2017. Grounding of HTN planning domain. *International Journal on Artificial Intelligence Tools* 26(05):1760021.

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 Twenty-Ninth International Conference on Automated Planning and Scheduling*, 382–390.

Schreiber, D. 2021. Lilotane: A lifted SAT-based approach to hierarchical planning. *Journal of Artificial Intelligence Research* 70:1117–1181.

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