A Simple Yet Efficient Lifted Formulation for Hard-to-Ground Planning Problems

Miquel Bofill
Departament d'Informàtica, Matemàtica Aplicada i Estadística, Universitat de Girona, Spain

Cristina Borralleras
Departament d'Enginyeries, Universitat de Vic - Universitat Central de Catalunya, Spain

Josu Oca
Departament d'Informàtica, Matemàtica Aplicada i Estadística, Universitat de Girona, Spain

Abstract

Planning domains are usually represented in first-order logic. However, historically, most of the planners have been using grounded problem representations. Since grounding can lead to a combinatorial explosion, and thus make some problems intractable, the community has developed techniques to deal with hard-to-ground problems. Those techniques include, for example, splitting action schemas, or partially grounding actions to relevant instantiations for the goal at hand. Recently, a new family of planners which work directly on lifted representations has emerged. However, most of these so-called *lifted planners* still rely on grounding to some extent. With the aim of broadening the range of approaches to lifted planning, in this work we present a satisfiability-based lifted planner that works on first-order representations and does not rely on grounding at all. We encode the lifted planning task into a quantifier-free formula with uninterpreted functions, in first-order logic with equality. Despite showing higher solving times than the state-of-the art, the planner proves to consume very little memory and is competitive on hard-to-ground instances.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Planning and scheduling → Planning for deterministic actions

Keywords and phrases Planning graphs, lifted planning, SMT

Funding This work was funded by MCIN/MICIU/AEI/10.13039/501100011033 and by ERDF A way of making Europe (grants PID2021-122274OB-I00, PRE2022-102047 and PID2024-157625OB-I00).

1 Introduction

The representation of planning problems is usually done in first-order logic. However, most of the planners work on grounded representations, i.e., on equivalent formulations where all predicates are instantiated with the available objects. The rationale for grounding is mainly the simplicity and efficiency of the techniques it allows.

The most prominent approaches to AI planning are based on heuristic search, where the initial state is expanded towards promising successor states, or on satisfiability testing, where propositional Boolean formulas are used to encode the existence of a plan of a certain length. In both cases, grounding can be a bottleneck, since it often results in a prohibitively large set of elements.

Several techniques have been developed to mitigate the effect of grounding. For example, the Fast Downward planner [7] overapproximates the ground predicates and actions that can be useful for finding a plan in state-space search using the delete-free relaxation of the problem. This sometimes removes from consideration a large number of ground elements. In the context of satisfiability-based planning, a well-known way to avoid full grounding is to split action schemas, as proposed in [15].

More recently, planners working on lifted representations have been developed. However, most of them do not perform deduction directly on the lifted representation, but rather some

kind of grounding on-demand during state-space search. There also exist satisfiability-based lifted planners. In this context, LiSAT [8] is state-of-the-art. This solver uses a state-less lifted representation of the problem, that is finally encoded into a propositional formula encoding a sequence of steps of a plan. On the other hand, Q-Planner [16] is a lifted planner encoding planning problems to quantified Boolean formulas (QBF). For a survey on *lifted planners*, see [5].

Satisfiability Modulo Theories (SMT) is the problem of solving first-order logic formulas with respect to background theories. SMT solvers typically consist of a SAT solver cooperating with several theory solvers in a modular way. In SMT we can have formulas combining, e.g., equality and uninterpreted functions, like $\neg f(x) \vee g(x) \vee (h(y, z) \wedge y = x)$. This is a natural extension to SAT, providing a considerable increase in expressiveness with little loss of performance in many cases.

In this paper we present a satisfiability-based lifted planner that encodes planning tasks into SMT formulas, maintaining a lifted representation also in solving time.

2 Planning Graphs

Since the inception of the Graphplan algorithm [3], many planners have been using planning graphs to guide their search. A *planning graph* is a layered directed graph, alternating proposition and action layers. Roughly, proposition layers contain the propositions that are possibly true at a given time step (e.g., the initial step), while action layers contain the actions whose preconditions are satisfied by propositions in the previous layer. Edges represent relations between actions and propositions. In the STRIPS setting, actions are connected by edges to their preconditions in the previous propositions layer, and to their effects in the next propositions layer (add and delete effects are differentiated using two kinds of edges). An important part of planning graph analysis is to deal with exclusion relationships between actions, i.e., to determine whether two actions cannot belong to the same plan.

Although planning graphs are mostly used in search-based planners, Kautz, McAllester, and Selman showed that they are quite similar to propositional formulas and, in fact, they can be automatically translated to SAT [12]. Therefore, they have also been used in some satisfiability-based planners [10, 11].

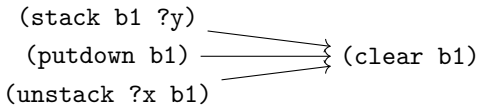
Planning graphs are typically built on grounded representations and, although they are not exactly state-space graphs, in the sense that not all possible instantiations of actions and facts are present in them, they tend to suffer from combinatorial explosion. To mitigate the combinatorial explosion and tackle hard-to-ground planning problems, we propose to use a *lifted planning graph*. The basic idea is to represent the planning problem with a planning graph with first-order predicates for actions and facts, and to encode that graph into a first order formula to be solved by an SMT solver instead of a SAT solver.

3 Lifted Planning on Quantifier-Free First-Order Formulas

In this section we explain how to build a lifted planning graph, for a plan of a certain length, and describe a possible translation of it into SMT. We restrict ourselves to the STRIPS formalism and to length optimal sequential plans, i.e., plans achieving the goal in the minimum possible number of steps.

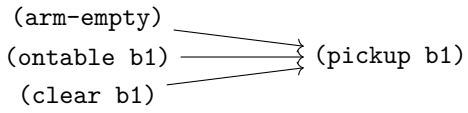
In [4] it is pointed out that the size of the graph can be unnecessarily large if there are many irrelevant facts in the initial conditions. This could be mitigated by beginning with a

regression analysis going backward from the goals to determine if any initial conditions may be thrown out, although this regression analysis can require some time. Making Graphplan goal directed, i.e., building it backwards from the goal, was first explored in [9]. The width of the graph will depend mainly on the branching factor. Forward construction will be problematic when there are many applicable actions, while backward construction will be problematic when there are many relevant actions. Nevertheless, we propose to build the graph backwards from the goal. The rationale is that this normally results in a smaller graph, if we keep arguments uninstantiated when possible. As an example, consider a typical blocks domain with `pickup`, `putdown`, `stack` and `unstack` operators. By going backwards, given a fact like `(clear b1)` we can generate the following edges from lifted actions having that fact as an add effect, where `?x` and `?y` are variables:



That is, block `b1` will become `clear` (if it was not already) either after stacking it on top of some other object, or putting it down, or unstacking any other object on top of it. There is no need to specify what these other objects are.

On the contrary, building a lifted graph going forward is not an easy task. In particular, from ground facts we are going to get ground actions. This is because, while an effect can be obtained in several ways, an action requires all preconditions to hold in order to be activated. Therefore, there will be no free variables in the activated action. For example:



3.1 From Lifted Planning Graphs to SMT Formulae

We build the planning graph recursively from the goal. We set the last propositions layer to contain the ground facts from the goal, and seek for action schemas having these facts as possible add effects. We add those lifted actions to the actions layer before the goal, and connect them to the goal facts with the corresponding variable bindings. Then we create a propositions layer preceding that actions layer, containing all their (lifted) preconditions, together with the previous goal, and proceed recursively taking them as the new goal. It is worth noting that goals are propagated to the previous layers, since we do not know in advance in which step they will be achieved (see Figure 1).

The graph is constructed for a given number of levels n. Then we proceed to translate the resulting graph into an SMT formula which will be satisfiable if and only if there exists a sequential plan of length n. Our planner will construct those graphs for $n = 0, 1, 2, \ldots$ and check for satisfiability of the resulting formulas in order to find a length optimal plan.

The translation goes as follows. Predicates are declared as Boolean functions with integer arguments. The ones that occur in effects, i.e., the fluents, are moreover indexed by time (see Listing 3, lines 4–19). Action schemas are declared as Boolean constants, indexed by time (Listing 3, lines 21–24). As we explain below, the concrete values of their arguments will be captured by frame axioms.

Integer variables are introduced for every predicate and action argument (Listing 3, lines 26–30). Variables for arguments of actions can be shared between actions, since we allow only one action to be executed at each time step.

We start by asserting the goal, i.e., the facts in the last propositions layer (Listing 3, lines 32–33). Moreover, for each propositions layer in the planning graph, we assert a set of frame axioms: for each proposition in the layer, either it was already true in the previous step of the plan or some of the actions having it as an add effect has been executed in the previous step, with the corresponding binding of variables (Listing 3, lines 38–40 and 57–63). It is worth noting that, since we are not considering negative preconditions (because we restrict ourselves to the STRIPS formalism), frame axioms explaining the reason for some fact becoming false are not necessary.

For each actions layer, we assert that exactly one action is executed (Listing 3, lines 42–45 and 65–69), and that their preconditions and effects are implied (Listing 3, lines 47–53 and 71–84). Variables are left uninstantiated here; note that they will be instantiated by the frame axioms.

Finally, we need to assert the initial facts. An important issue is to avoid combinatorial explosion due to the closed world assumption. Naively asserting the negation of any fact which does not hold initially can represent 99% of the entire formula in some hard-to-ground domains. So, instead of asserting all those negated facts, we only negate the ground facts that occur in the first propositions layer and do not hold initially. For non-ground predicates, we constrain the values of their variables to the ones in the initial facts, or negate the predicate in case no initial fact is an instance of them (Listing 3, lines 86–98).

3.2 Example

We illustrate our approach with a toy example. Consider the following simplified domain.

Listing 1 `blocksworld` domain with `putdown` and `unstack` operators removed.

```
(define (domain blocksworld)
(:requirements :strips)
(:predicates (clear ?x)
             (on-table ?x)
             (arm-empty)
             (holding ?x)
             (on ?x ?y))

(:action pickup
  :parameters (?x)
  :precondition (and (clear ?x) (on-table ?x) (arm-empty))
  :effect (and (holding ?x) (not (clear ?x)) (not (on-table ?x))
               (not (arm-empty))))

(:action stack
  :parameters (?x ?y)
  :precondition (and (clear ?y) (holding ?x))
  :effect (and (arm-empty) (clear ?x) (on ?x ?y) (not (clear ?y))
               (not (holding ?x)))))
```

Consider the following toy problem on this domain.

Listing 2 A toy problem on the `blocksworld` domain.

```
(define (problem blocks-toy)
```

```
(:domain blocksworld)

(:objects b1 b2 b3)
(:init
  (arm-empty)
  (on-table b1)
  (clear b1)
  (on-table b2)
  (clear b2)
  (on-table b3)
  (clear b3)
)
(:goal (and
  (on b2 b1)
)))
```

Figure 1 shows the lifted planning graph built backwards from the previous toy example, for a 2-steps plan.

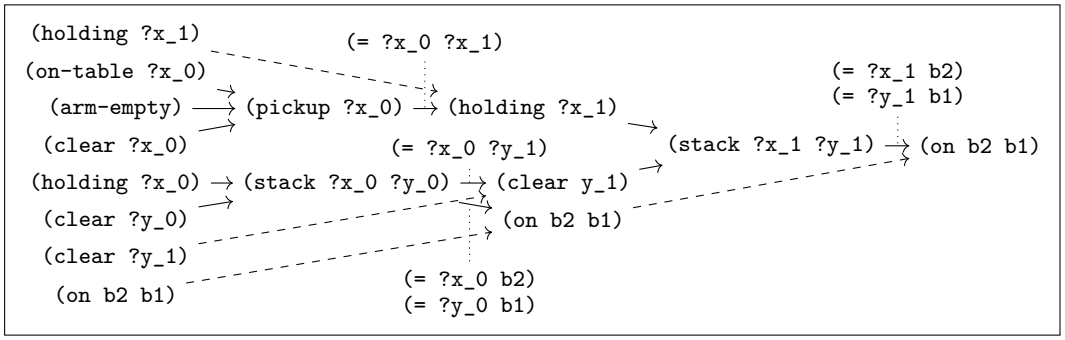


Figure 1 Planning graph with 2 layers resulting from the previous `blocksworld` problem. Dashed lines represent implicit *maintain* operations. Equality constraints capture the necessary bindings.

The SMT formula for the planning graph of Figure 1 would be the following.

Listing 3 Example SMT formula.

```
(set-logic QF_UFIDL)
(set-option :produce-models true)

;; Predicates
(declare-fun clear_0 (Int) Bool)
(declare-fun clear_1 (Int) Bool)
(declare-fun clear_2 (Int) Bool)
(declare-fun on-table_0 (Int) Bool)
(declare-fun on-table_1 (Int) Bool)
(declare-fun on-table_2 (Int) Bool)
(declare-fun arm-empty_0 () Bool)
(declare-fun arm-empty_1 () Bool)
(declare-fun arm-empty_2 () Bool)
(declare-fun holding_0 (Int) Bool)
(declare-fun holding_1 (Int) Bool)
(declare-fun holding_2 (Int) Bool)
(declare-fun on_0 (Int Int) Bool)
```

```
(declare-fun on_1 (Int Int) Bool)
(declare-fun on_2 (Int Int) Bool)

;; Actions variables
(declare-const _pickup_?x_0 Bool)
(declare-const _stack_?x_?y_0 Bool)
(declare-const _stack_?x_?y_1 Bool)

;; Objects' variables
(declare-const ?x_1 Int)
(declare-const ?x_0 Int)
(declare-const ?y_1 Int)
(declare-const ?y_0 Int)

;; Goal
(assert (and (on_2 2 1)))

;; Planning graph
;; Level 1

;; Frame axioms
(assert (or (not (on_2 2 1)) (on_1 2 1)
            (and _stack_?x_?y_1 (= ?x_1 2) (= ?y_1 1))))

;; At most one action per level

;; At least one action per level
(assert (or _stack_?x_?y_1))

;; Actions preconditions and effects
(assert (=> _stack_?x_?y_1
            (and (holding_1 ?x_1) (clear_1 ?y_1))))
(assert (=> _stack_?x_?y_1
            (and (on_2 ?x_1 ?y_1) arm-empty_2 (clear_2 ?x_1))))
(assert (=> _stack_?x_?y_1
            (and (not (holding_2 ?x_1)) (not (clear_2 ?y_1)))))

;; Level 0

;; Frame axioms
(assert (or (not (on_1 2 1)) (on_0 2 1)
            (and _stack_?x_?y_0 (= ?x_0 2) (= ?y_0 1))))
(assert (or (not (holding_1 ?x_1)) (holding_0 ?x_1)
            (and _pickup_?x_0 (= ?x_0 ?x_1))))
(assert (or (not (clear_1 ?y_1)) (clear_0 ?y_1)
            (and _stack_?x_?y_0 (= ?x_0 ?y_1))))

;; At most one action per level
(assert (not (and _pickup_?x_0 _stack_?x_?y_0)))

;; At least one action per level
(assert (or _pickup_?x_0 _stack_?x_?y_0))

;; Actions preconditions and effects
(assert (=> _pickup_?x_0
```

```
            (and (on-table_0 ?x_0) arm-empty_0 (clear_0 ?x_0))))
(assert (=> _pickup_?x_0
            (holding_1 ?x_0)))
(assert (=> _pickup_?x_0
            (and (not (on-table_1 ?x_0)) (not arm-empty_1)
                 (not (clear_1 ?x_0)))))
(assert (=> _stack_?x_?y_0
            (and (holding_0 ?x_0) (clear_0 ?y_0))))
(assert (=> _stack_?x_?y_0
            (and (on_1 ?x_0 ?y_0) arm-empty_1 (clear_1 ?x_0))))
(assert (=> _stack_?x_?y_0
            (and (not (holding_1 ?x_0)) (not (clear_1 ?y_0)))))

;; Lifted initial state
(assert (not (holding_0 ?x_1)))
(assert (or (not (on-table_0 ?x_0))
            (= ?x_0 1) (= ?x_0 2) (= ?x_0 3)))
(assert arm-empty_0)
(assert (or (not (clear_0 ?x_0))
            (= ?x_0 1) (= ?x_0 2) (= ?x_0 3)))
(assert (not (holding_0 ?x_0)))
(assert (or (not (clear_0 ?y_0))
            (= ?y_0 1) (= ?y_0 2) (= ?y_0 3)))
(assert (or (not (clear_0 ?y_1))
            (= ?y_1 1) (= ?y_1 2) (= ?y_1 3)))
(assert (not (on_0 2 1)))
(check-sat)
(get-model)
(exit)
```

Note that we are using quantifier-free logic with integers and uninterpreted function symbols, so free variables are unbound placeholders that can take any value. However, they are conveniently restricted by the constraints we add for the lifted initial state.

3.3 Additional Considerations

Modern SMT solvers add support for datatypes, including enumerated types. Therefore, integer variables could be replaced by enumerated types. However, we observed that doing so slightly degrades performance. Another alternative could be using uninterpreted sorts. But, in our context, using uninterpreted sorts has the problem of having to state explicitly that all values of each sort are different which, in case of problems with many objects, can cause a big overhead in the formula. On the contrary, all integer values are different by definition.

On the other hand, there exist some domains which include static predicates, i.e., predicates that occur in preconditions but not in effects. Those predicates are typically used for typing, and can be deleted in a preprocessing step. By converting the names of those predicates into types of the qualified objects, we can obtain an equivalent but simpler formula.

4 Experimental Evaluation

We report on experiments on the compilation of hard-to-ground domains available at `https://github.com/abcorrea/htg-domains`. The experiments were conducted on an Intel Xeon

E-2234 CPU@3.60GHz, with a memory limit of 16GB and a time limit of 1 hour. For the SMT solver we used CVC5 [1] v1.3.0 with CaDiCaL [2] v3.0.0. We compare our *domain-Agnostic Lifted Planning System* ALPS against the state-of-the-art satisfiability-based lifted planner LiSAT [8], the QBF-based planner without grounding Q-Planner [16], the state-of-the-art satisfiability-based grounded planner Madagascar [14] without invariant computation nor parallelism, and the state-of-the-art grounded heuristic planner Fast Downward [6].

As said in Section 3.3, some domains include static predicates. For example, in the `logistics-large-simple` domain, we have the `truck`, `location`, `airplane`, `city`, and `airport` unary predicates, used for typing. Similarly, the `pipesworld-tankage-nosplit` domain includes the `unitary` and `not-unitary` static unary predicates. Removing those predicates and adding the predicate names as types of the qualified objects yielded shorter solving times.

Table 1 shows the coverage results for the considered planners on the aforementioned benchmarks. We exclude the domains with negative preconditions and equality, as we currently do not support them, as well as the domains that no planner was able to solve with the given memory and time limits. We observe that the coverage of our approximation is not far from that of LiSAT. It is worth noting that LiSAT translates the lifted planning problem into SAT and performs several low-level optimizations, while we keep solving an SMT formula, with the corresponding overhead in execution time. ALPS's results are quite good, considering the simplicity of the encoding and the technology used.

Table 1 Number of solved instances.

	ALPS	LiSAT	Q-Planner	Madagascar	Fast Downward
blocksworld-large-simple (40)	**40**	**40**	1	3	12
childsnack-contents (144)	12	**24**	8	1	12
logistics-large-simple (40)	29	**35**	0	0	21
pipesworld-tankage-nosplit (50)	19	**25**	0	6	11
visitall-multidimensional (180)	99	**111**	10	40	78
Coverage (454)	199	**235**	19	50	134

5 Future Work

As future work, we plan to add support for negative preconditions as well as equality predicates. In terms of performance, a possible improvement could be to use the CVC5 API in order to solve the problem incrementally, instead of generating independently solved formulas for each plan horizon, as we currently do. Also, a major challenge is to adapt our approach for parallel plans, since the performance of our approach degrades rapidly with plan length. On the other hand, since SMT is well suited to numeric plans, it is also worth considering an extension of our approach to numeric planning. The translation of our lifted formulas to SAT could also be explored. Finally, the constraints related to our lifted initial state are essentially *table* constraints. Therefore, hybrid SMT-CP or full CP approaches could be investigated. In fact, given the MiniZinc [13] support for equality and logical operators, it would be worth exploring MiniZinc models similar to our formulation.

References

1 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A Versatile and Industrial-Strength SMT Solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. doi:10.1007/978-3-030-99524-9_24.

2 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

3 Avrim Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1636–1642. Morgan Kaufmann, 1995. URL: http://ijcai.org/Proceedings/95-2/Papers/080.pdf.

4 Avrim Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. *Artif. Intell.*, 90(1-2):281–300, 1997. doi:10.1016/S0004-3702(96)00047-1.

5 Augusto B. Corrêa and Giuseppe De Giacomo. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In Kate Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 8010–8019. International Joint Conferences on Artificial Intelligence Organization, 8 2024. Survey Track. doi:10.24963/ijcai.2024/886.

6 Malte Helmert. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26:191–246, 2006. URL: https://doi.org/10.1613/jair.1705, doi:10.1613/JAIR.1705.

7 Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5):503–535, 2009. Advances in Automated Plan Generation. URL: https://www.sciencedirect.com/science/article/pii/S0004370208001926, doi:10.1016/j.artint.2008.10.013.

8 Daniel Höller and Gregor Behnke. Encoding Lifted Classical Planning in Propositional Logic. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 134–144. AAAI Press, 2022.

9 Subbarao Kambhampati, Eric Parker, and Eric Lambrecht. Understanding and extending Graphplan. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning*, pages 260–272, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

10 Henry Kautz and Bart Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *AIPS98 workshop on planning as combinatorial search*, volume 58260, pages 58–60. Carnegie Mellon University Pittsburgh, Penn., 1998.

11 Henry Kautz, Bart Selman, and Jörg Hoffmann. SatPlan: Planning as satisfiability. In *5th international planning competition*, volume 20, page 156, 2006.

12 Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding Plans in Propositional Logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996.*, pages 374–384. Morgan Kaufmann, 1996.

13 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

14 Jussi Rintanen. Madagascar: Scalable planning with SAT. *Proceedings of the 8th International Planning Competition (IPC-2014)*, 21:1–5, 2014.

15 Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. SAT-Based Parallel Planning Using a Split Representation of Actions. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI, 2009. URL: `http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/732`.

16 Irfansha Shaik and Jaco van de Pol. Classical Planning as QBF without Grounding. In Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 329–337. AAAI Press, 2022. URL: `https://ojs.aaai.org/index.php/ICAPS/article/view/19817`.