Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Classical Planning as QBF without Grounding

Irfansha Shaik, Jaco van de Pol

Aarhus University, Department of Computer Science, Aarhus, Denmark
irfansha.shaik@cs.au.dk, jaco@cs.au.dk

Abstract

Most classical planners use grounding as a preprocessing step, essentially reducing planning to propositional logic. However, grounding involves instantiating all action rules with concrete object combinations, and results in large encodings for SAT/QBF-based planners. This severe cost in memory becomes a main bottleneck when actions have many parameters, such as in the Organic Synthesis problems from the IPC 2018 competition. We provide a compact QBF encoding that is logarithmic in the number of objects and avoids grounding completely, by using universal quantification for object combinations. We show that we can solve some of the Organic Synthesis problems, which could not be handled before by any SAT/QBF based planners due to grounding.

Introduction

Automated planning has many real-world applications, such as Space Exploration and Robotics, cf. the book by Ghallab, Nau, and Traverso (2004). The three main research directions in automated planning are heuristic-based state-space search, symbolic search and propositional satisfiability (SAT) based solving. While heuristic-based search often finds some plan quickly, it may not guarantee to search the whole search space. SAT-based solvers, on the other hand, can also be used to prove the non-existence of plans up to a bounded length and finding optimal plans. For some applications, quickly falsifying the existence of a plan or finding provably optimal plans can be useful. Classical planning is the most simple problem; its aim is to find a valid sequence of actions from a single initial state to some goal state, where the state is completely known and the effect of all actions is deterministic. Kautz and Selman (1992) reduced the planning problem to the bounded reachability problem and provided a corresponding SAT encoding for a plan of length k.

Classical planning domains are usually defined by logical rules that describe the pre-conditions and effects of applying actions. Usually, a single action involves multiple objects, corresponding to the arity of that action. The pre-conditions and effects contain predicates, which also refer to multiple objects. A concrete planning problem defines a universe of concrete objects, and an initial and goal condition. Many planning tools, both SAT-based and heuristic, first apply a grounding step: the action rules are instantiated for all combinations of objects. However, for domains with actions involving many objects, the grounded specification is very large, so memory becomes the main bottleneck. Despite several improvements, such as action splitting (Kautz and Selman 1992), explanatory frame axioms (Haas 1987), invariants (Rintanen 2008) and parallel plans (Rintanen, Heljanko, and Niemelä 2006; Robinson et al. 2009), the memory still remains a bottleneck for domains with large action arity.

QBF (Quantified Boolean Formula) encodings are known for being more compact than SAT encodings, so they are considered as an alternative when SAT encodings suffer from memory problems. Dershowitz, Hanna, and Katz (2005) and Jussila and Biere (2007) proposed a QBF encoding with $\exists\forall\exists$ quantifier alternation for reachability in Bounded Model Checking (BMC). It generates only a single copy of the transition function, instead of k copies in SAT, by using quantification over state variables. Rintanen (2001) proposed a reachability QBF encoding that is logarithmic in the length of the plan, and uses only one transition function. Cashmore, Fox, and Giunchiglia (2012) proposed the Compact Tree Encoding (CTE), which improves upon the logarithmic encoding by Rintanen (in the context of planning) by efficient traversal of the search tree. Although these QBF encodings are more concise than the SAT encodings, it has been reported that the SAT encodings can usually be solved faster than the QBF encodings by the current solvers (Cashmore, Fox, and Giunchiglia 2012). More importantly, the QBF encodings mentioned above still require the problem to be grounded first, so the memory bottleneck for actions that involve many objects has not been solved.

Matloob and Soutchanski (2016) proposed Organic Synthesis benchmarks and showed that SAT encodings could not solve any of them. This is consistent with the findings from the IPC-2018 planning competition, in which SAT based planners performed poorly on instances from Organic Synthesis. These benchmarks contain actions that manipulate up to 31 objects, so the grounding step exhausts the memory. Even heuristic planners that employ grounding will exhaust the memory for this domain. Thus, to solve these problems, there is a need for encodings that avoid grounding altogether.

Our Contribution. In this paper, we propose a QBF encoding of quantifier structure $\exists\forall\exists$ which completely avoids

grounding, by using universal variables to represent combinations of objects. The encoding grows linearly in the number of action names, predicates, and path length, and logarithmic in the number of objects. We provide an open-source implementation[1] of the encoding. We evaluate our QBF encoding on 18 domains from previous IPC competitions and on 10 hard-to-ground domains, and try to solve them with QBF solver CAQE. We compare the size of the encoding and the time needed for solving them with the best SAT-based solver Madagascar, and two of the best heuristic solvers.

Preliminaries

We introduce the main notions of the classical planning problems and QBF. We also present a running example of the blocks-world domain (Listing 1) in PDDL, the Planning Domain Definition Language (Fox and Long 2003).

Classical Planning

Definition 1 *A* planning signature $\Sigma = \langle \mathrm{P}, \mathrm{A}, \mathrm{O}, \mathrm{arity} \rangle$ *is a* 4*-tuple where* P *is a set of predicate symbols,* A *is a set of action names,* O *is a set of objects, and the function* $\mathrm{arity} : (\mathrm{P} \cup \mathrm{A} \to \mathbb{N}_0)$ *indicates the expected number of arguments.*

In the example of Listing 1, $\mathrm{P} = \{\mathrm{clear}, \mathrm{ontable}, \mathrm{on}\}$, $\mathrm{O} = \{b_1, b_2\}$, $\mathrm{A} = \{\mathrm{unstack}, \mathrm{stack}\}$, and $\mathrm{arity}(\mathrm{on}) = 2$. A schematic action corresponds to an action name with parameters, e.g. $\mathrm{stack}(?x_1, ?x_2)$. It can be grounded to a set of concrete actions, by replacing the parameters by concrete object combinations. Predicate symbols are also equipped with parameters, that can be grounded by concrete objects. To facilitate our encodings, we unify the parameter names of all actions to a fixed sequence of x's, and the formal predicate parameters to a sequence of y's. Note that adding dummy parameters does not change the problem. The fixed predicate parameter names will be used later for universal object combination variables in the QBF encoding. Since we use a sequential plan semantics, a single set of action and predicate parameter variables is sufficient to represent any chosen action at each time step. Each action is specified schematically by preconditions and effects.

Definition 2 *Given a signature, we define the set of atoms and grounded atoms (or fluents) as:*

$$\mathrm{Atom} = \{\mathrm{p}(\overrightarrow{x}) \mid \mathrm{p} \in \mathrm{P}, |\overrightarrow{x}| = \mathrm{arity}(\mathrm{p})\}$$
$$\mathrm{F} = \{\mathrm{p}(\overrightarrow{\mathrm{o}}) \mid \mathrm{p} \in \mathrm{P}, \mathrm{o}_i \in \mathrm{O}, |\overrightarrow{\mathrm{o}}| = \mathrm{arity}(\mathrm{p})\}$$

Definition 3 *Given a signature* $\Sigma = \langle \mathrm{P}, \mathrm{A}, \mathrm{O}, \mathrm{arity} \rangle$*. The planning problem* $\Pi = \langle \mathrm{I}, \mathrm{G}, \mathrm{pre}^+, \mathrm{pre}^-, \mathrm{eff}^+, \mathrm{eff}^- \rangle$ *is a* 6*-tuple where*

- *Initial state* $\mathrm{I} \subseteq \mathrm{F}$ *and*
- *Goal condition* $\mathrm{G} = (g^+, g^-)$*, where* $g^+, g^- \subseteq \mathrm{F}$.
- *for each action* $\mathrm{a} \in \mathrm{A}$*, we have:*
  - *positive preconditions* $\mathrm{pre}^+(\mathrm{a}) \subseteq \mathrm{Atom}$
  - *negative preconditions* $\mathrm{pre}^-(\mathrm{a}) \subseteq \mathrm{Atom}$
  - *positive effects* $\mathrm{eff}^+(\mathrm{a}) \subseteq \mathrm{Atom}$
  - *negative effects* $\mathrm{eff}^-(\mathrm{a}) \subseteq \mathrm{Atom}$

[1] https://github.com/irfansha/Q-Planner

Listing 1: blocks-world domain and problem

```
1  (define (domain blocksworld)
2  (:predicates (clear ?y1)(ontable ?y1)
3        (on ?y1 ?y2))
4  (:action unstack
5   :parameters (?x1 ?x2)
6   :precondition (and (clear ?x1)
7        (on ?x1 ?x2))
8   :effect (and (not (on ?x1 ?x2))
9        (ontable ?x1)(clear ?x2)))
10 (:action stack
11  :parameters (?x1 ?x2)
12  :precondition (and (clear ?x1)
13       (clear ?x2)(ontable ?x1))
14  :effect (and (not(clear ?x2))
15       (not(ontable ?x1))(on ?x1 ?x2))))
16 (define (problem BW_rand_2)
17 (:domain blocksworld)
18 (:objects b1 b2)
19 (:init (ontable b1) (on b2 b1) (clear b2))
20 (:goal (and (on b1 b2))))
```

In the example, (on $?x_1$ $?x_2$) is a positive precondition for action `unstack`, while (ontable $?x_1$) is a negative effect for action `stack`. We assume that the variables used in preconditions and effects are (non-dummy) action parameters.

SAT Encoding

The planning problem is encoded as finding a path (of length k) in a graph, whose nodes consist of assignments to the fluents and whose edges are defined by the conditions and effects of grounded actions. Usually, a SAT encoding uses propositional variables encoding the k actions and $k+1$ states in the path. A direct encoding would use k variables for each grounded action, but since we use sequential plan semantics (in which only one action occurs at the time), this can be encoded by a number of variables that is logarithmic in the number of grounded actions (still linear in k).

We will present the constraints in circuit format (with nested $\wedge$ and $\vee$), which can be transformed to CNF clauses by a standard procedure. We will write $\bigwedge$ and $\bigvee$ for conjunctions and disjunctions over finite (indexed) sets.

Quantified Boolean Formulas

Quantified Boolean Formulas (QBF) extend propositional formulas in SAT with universal quantification over some variables. A QBF formula in *prenex normal form* is of the shape $\Pi\phi$ where ϕ is called the matrix, which is a propositional formula, and $\Pi = Q_1 X_1 \ldots Q_n X_n$ is the prefix with alternating existential and universal quantifiers $Q_i \in \{\forall, \exists\}$ and disjoint sets of variables X_i. The Q_i specify the quantification of each variable that occurs in ϕ. We consider only closed formulas, i.e., the quantification of all variables is known. A QBF is evaluated to either True or False and its truth value can be computed by recursively solving the formula over each outermost variable. A formula $\exists x\phi$ is true if and only if $\phi[\top/x]$ or $\phi[\bot/x]$ is true. A formula $\forall x\phi$ is true if and only if both $\phi[\top/x]$ and $\phi[\bot/x]$ are true. The order of the variables within each quantified block does not matter.

Intermediate SAT Encoding

We now provide the intermediate SAT encoding of planning problems, as an introduction to our QBF encoding. Throughout the section, we assume a fixed planning problem as in Definition 3. Recall that we introduced a fixed sequence of action parameters x.

For the encoding, we generate copies of variables for each time step (represented by superscript) for a path of length k. We use action variables $\mathrm{A}^i = \{\mathrm{a}^i_b \mid 1 \leq b \leq \sigma\}$ for each $0 \leq i \leq k-1$ and $\sigma = \lceil \log(|\,\mathrm{A}\,|) \rceil$; here a^i_b represents the b-th bit of a logarithmic encoding of the action name that occurs at time i in the plan. The action parameter variables are $\mathrm{PM}^i = \{\mathrm{x}^i_{j,b} \mid 1 \leq j \leq \zeta, 1 \leq b \leq \gamma\}$ for each $0 \leq i \leq k-1$, where ζ is the maximum action arity and $\gamma = \lceil \log(|\,\mathrm{O}\,|) \rceil$. Here $\mathrm{x}^i_{j,b}$ represents the b-th bit of parameter x_j of the action scheduled at time i in the plan. We will write $\overrightarrow{\mathrm{x}^i_j}$ (sequence of γ variables) for the parameter x_j at time i. The state variables are $\mathrm{F}^i = \{\mathrm{f}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})} \mid \mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{F}\}$, i.e., one variable for every fluent and for each $0 \leq i \leq k$.

The corresponding SAT encoding is based on these variables, and constraints for the initial condition I, goal condition G, transition function $\mathrm{T_g}$, and a domain restriction RC.

$$\exists\, \mathrm{A}^0, \mathrm{PM}^0, \ldots, \mathrm{A}^{k-1}, \mathrm{PM}^{k-1}\, \exists \mathrm{F}^0, \ldots, \mathrm{F}^k$$
$$\mathrm{I}(\mathrm{F}^0) \wedge \mathrm{G}(\mathrm{F}^k) \wedge \bigwedge_{i=0}^{k-1} \mathrm{T}^i_{\mathrm{g}}(\mathrm{F}^i, \mathrm{F}^{i+1}, \mathrm{A}^i, \mathrm{PM}^i) \wedge$$
$$\bigwedge_{i=0}^{k-1} \mathrm{RC}(\mathrm{A}^i, \mathrm{PM}^i)$$

In the constraint for the initial state, each variable is positive if the proposition is in the set I and negative if it is not. For the goal state, the variable is positive if the proposition is in g^+ and negative if it is in g^-. Note that this provides a unique initial state, but there may be multiple states that satisfy the goal condition. We group the constraints (throughout the article) based on the predicates for the sake of easy understanding of the correctness proof.

Definition 4 *Initial constraint* $\mathrm{I}(\mathrm{F}^0) =$

$$\bigwedge_{\mathrm{p}\in\mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}})\in\mathrm{F}} \{\mathrm{f}^0_{\mathrm{p}(\overrightarrow{\mathrm{o}})} \,|\, \mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{I}\} \wedge$$
$$\bigwedge_{\mathrm{p}\in\mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}})\in\mathrm{F}} \{\neg\, \mathrm{f}^0_{\mathrm{p}(\overrightarrow{\mathrm{o}})} \,|\, \mathrm{p}(\overrightarrow{\mathrm{o}}) \notin \mathrm{I}\}$$

Definition 5 *Goal constraint* $\mathrm{G}(\mathrm{F}^k) =$

$$\bigwedge_{\mathrm{p}\in\mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}})\in\mathrm{F}} \{\mathrm{f}^k_{\mathrm{p}(\overrightarrow{\mathrm{o}})} \,|\, \mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{g}^+\} \wedge$$
$$\bigwedge_{\mathrm{p}\in\mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}})\in\mathrm{F}} \{\neg\, \mathrm{f}^k_{\mathrm{p}(\overrightarrow{\mathrm{o}})} \,|\, \mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{g}^-\}$$

To encode the transition function, we will generate five constraints for each proposition variable. These constraints define the value of all fluents and time stamps, just before or after some action occurs. The first four constraints correspond to the positive preconditions, negative preconditions, positive effects and negative effects, respectively. The last constraint corresponds to the frame axiom, which indicates that untouched propositions should not change.

Definition 6 *The **grounded transition function** is:*

$$\mathrm{T}^i_{\mathrm{g}}(\mathrm{F}^i, \mathrm{F}^{i+1}, \mathrm{A}^i, \mathrm{PM}^i) = \bigwedge_{\mathrm{p}\in\mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}})\in\mathrm{F}} \mathrm{PC}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})}$$

*where the **Proposition Constraint** $\mathrm{PC}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})}$ =*

$$(\mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{pre}^+) \implies \mathrm{f}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})}) \wedge$$
$$(\mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{pre}^-) \implies \neg \mathrm{f}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})}) \wedge$$
$$(\mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{eff}^+) \implies \mathrm{f}^{i+1}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}) \wedge$$
$$(\mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{eff}^-) \implies \neg \mathrm{f}^{i+1}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}) \wedge$$
$$((\mathrm{f}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})} = \mathrm{f}^{i+1}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}) \vee \mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{eff}^+) \vee$$
$$\mathrm{AC}_{\mathrm{p}(\overrightarrow{\mathrm{o}})}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{eff}^-))$$

Consider the positive precondition constraint: in plain words it expresses that if some action occurs at time step i, and some fluent (grounded predicate) is in the positive precondition of that action, then the corresponding variable is true at time step i. Similarly, the proposition variables corresponding to the positive (negative) effects should be set to true (false) at time step $i+1$. The last constraint encodes the frame axiom: it expresses that either the value of a proposition variable stays the same, or some positive or negative effect occurs that defines this proposition.

We still need to encode when a positive/negative condition/effect Φ is associated with the current action. We also need to define the set of grounded instances of a predicate.

Definition 7 *Given an action* $\mathrm{a} \in \mathrm{A}$*, a set of atoms* $\Phi \in \{\mathrm{pre}^+, \mathrm{pre}^-, \mathrm{eff}^+, \mathrm{eff}^-\}$*, and sequence of objects* $\overrightarrow{\mathrm{o}} \in \mathrm{O}^n$ *with* $n = \mathrm{arity}(\mathrm{a})$*, its **grounding*** $\mathrm{GD}(\Phi, \mathrm{a}, \overrightarrow{\mathrm{o}}) \subseteq \mathrm{F}$ *is a set of fluents, defined* $\mathrm{GD}(\Phi, \mathrm{a}, \overrightarrow{\mathrm{o}}) := \{\phi[\overrightarrow{x}/\overrightarrow{\mathrm{o}}] \mid \phi \in \Phi(\mathrm{a})\}$.

The grounding of an action with given object parameters returns the corresponding grounded preconditions/effects. In the running example, the positive precondition of action stack for object parameters (b_1, b_2) is grounded as $\mathrm{GD}(\mathrm{pre}^+, \mathrm{stack}, (b_1, b_2)) = \{\mathrm{clear}(b_1), \mathrm{clear}(b_2), \mathrm{ontable}(b_1)\}$.

Below, we use "bin" to express the logarithmic encoding of objects by γ bits and of action names by σ bits. We also use "$=$" to denote a bit-wise conjunction of bi-implications.

Definition 8 *Given the set of action names* A *and a set of atoms* $\Phi \in \{\mathrm{pre}^+, \mathrm{pre}^-, \mathrm{eff}^+, \mathrm{eff}^-\}$ *and a fluent* f*, the **grounded action constraint*** $\mathrm{AC_f}(\mathrm{A}^i, \mathrm{PM}^i, \Phi) =$

$$\bigvee_{\mathrm{a}\in\mathrm{A}} \bigvee_{\substack{n=\mathrm{arity}(a) \\ \overrightarrow{\mathrm{o}}\in\mathrm{O}^n \text{ s.t.} \\ \mathrm{f}\in\mathrm{GD}(\Phi,\mathrm{a},\overrightarrow{\mathrm{o}})}} (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{a})) \wedge \bigwedge_{j=1}^{n} (\overrightarrow{\mathrm{x}^i_j} = \mathrm{bin}(\mathrm{o}_j))$$

In the grounded action constraint, for each grounded action $\mathrm{a}(\overrightarrow{\mathrm{o}})$ that contains fluent f in its grounded precondition/effect Φ, we generate equality constraints for action and parameter variables. For example, for fluent $\mathrm{clear}(b_1)$, $\Phi = \mathrm{pre}^+$ and grounded action $\mathrm{stack}(b_1, b_2)$, we generate a constraint $\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \mathrm{bin}(b_1) \wedge \overrightarrow{x_2^i} = \mathrm{bin}(b_2)$.

Finally, due to the use of logarithmic variables, invalid actions are possible when the number of objects and actions are non-powers of 2. To maintain consistency we restrict the invalid action and parameter values by the RC-constraint. Here $<$ denotes the bit-blasted comparison operator on binary numbers.

Definition 9 *Restricted constraints* $\mathrm{RC}(\mathrm{A}^i, \mathrm{PM}^i)$ =

$$\big(\overrightarrow{\mathrm{A}^i} < |\,\mathrm{A}\,|\big) \wedge \bigwedge_{1 \le j \le \zeta} \big(\overrightarrow{\mathrm{x}_j^i} < |\,\mathrm{O}\,|\big)$$

In practice, this constraint can still be refined if one has type information on the objects, as in typed PDDL domains.

New Ungrounded QBF Encoding

Similar to the intermediate SAT encoding, the QBF encoding of a plan of length k corresponds to finding a path of length k from the initial state to some goal state. The SAT encoding used propositional variables for each object-instance of predicate symbols and each time step. Note that the values of these propositional variables only depend on the action variables, i.e., given a sequence of actions and (complete) initial state, the intermediate states until the final state are completely determined. From the constraints on the propositional variables, one can also observe that their values are independent of each other, so these constraints can be enforced independently, at each time step and for each action.

For the QBF encoding, the idea is to avoid grounding by representing all object combinations as universal variables and generate constraints on the predicate variables directly. Essentially the universal object combination variables and existential predicate variables replace the existential propositional variables in the intermediate SAT encoding.

In the sequel, we assume a fixed planning problem as in Definition 3, and a fixed planning signature as in Definition 1. Recall that we introduced fixed variables x for formal action parameters and fixed variables y for formal predicate variables. These variables are used in the transformation.

The QBF encoding will use the same action variables (A) and action parameter variables (PM) as the SAT encoding. We define object combination variables and predicate variables separately. The object combination variables are $\mathrm{OC} = \{\mathrm{y}_{j,b} \mid 1 \le j \le \eta, 1 \le b \le \gamma\}$, where η is the maximum arity of predicates and γ is as defined before. Here, $\mathrm{y}_{j,b}$ represents the b-th bit of the jth object variable and $\overrightarrow{\mathrm{y}_j}$ (sequence of γ variables) represents the jth object variable. The (*ungrounded*) predicate variables are $\mathrm{P}^i = \{\mathrm{q}_\mathrm{p}^i \mid \mathrm{p} \in \mathrm{P}\}$ for each $0 \le i \le k$.

Prefix:

$\exists \overrightarrow{\mathrm{A}^0} \exists \overrightarrow{\mathrm{x}_1^0} \exists \overrightarrow{\mathrm{x}_2^0} \;\; \exists \overrightarrow{\mathrm{A}^1} \exists \overrightarrow{\mathrm{x}_1^1} \exists \overrightarrow{\mathrm{x}_2^1}$
$\forall \overrightarrow{y_1} \forall \overrightarrow{y_2}$
$\exists\, \mathrm{q}^0_{\mathrm{clear}} \exists\, \mathrm{q}^0_{\mathrm{ontable}} \exists\, \mathrm{q}^0_{\mathrm{on}}$
$\exists\, \mathrm{q}^1_{\mathrm{clear}} \exists\, \mathrm{q}^1_{\mathrm{ontable}} \exists\, \mathrm{q}^1_{\mathrm{on}}$
$\exists\, \mathrm{q}^2_{\mathrm{clear}} \exists\, \mathrm{q}^2_{\mathrm{ontable}} \exists\, \mathrm{q}^2_{\mathrm{on}}$

Initial state:

$((\mathrm{bin}(b_2) = \overrightarrow{y_1}) \iff \mathrm{q}^0_{\mathrm{clear}}) \wedge$
$((\mathrm{bin}(b_1) = \overrightarrow{y_1}) \iff \mathrm{q}^0_{\mathrm{ontable}}) \wedge$
$((\mathrm{bin}(b_2) = \overrightarrow{y_1}) \wedge (\mathrm{bin}(b_1) = \overrightarrow{y_2}) \iff \mathrm{q}^0_{\mathrm{on}}) \wedge$

Goal state:

$((\mathrm{bin}(b_1) = \overrightarrow{y_1}) \wedge (\mathrm{bin}(b_2) = \overrightarrow{y_2})) \implies \mathrm{q}^2_{\mathrm{on}} \wedge$

For time steps i = 0,1:

clear:

$\big((\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge (\overrightarrow{x_1^i} = \overrightarrow{y_1} \vee \overrightarrow{x_2^i} = \overrightarrow{y_1})) \vee$
$\quad (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1})\big) \implies \mathrm{q}^i_{\mathrm{clear}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_2^i} = \overrightarrow{y_1}) \implies \mathrm{q}^{i+1}_{\mathrm{clear}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_2^i} = \overrightarrow{y_1}) \implies \neg\, \mathrm{q}^{i+1}_{\mathrm{clear}} \wedge$
$((\mathrm{q}^i_{\mathrm{clear}} = \mathrm{q}^{i+1}_{\mathrm{clear}}) \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_2^i} = \overrightarrow{y_1})$
$\quad \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_2^i} = \overrightarrow{y_1})) \wedge$

ontable:

$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1}) \implies \mathrm{q}^i_{\mathrm{ontable}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1}) \implies \mathrm{q}^{i+1}_{\mathrm{ontable}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1}) \implies \neg\, \mathrm{q}^{i+1}_{\mathrm{ontable}} \wedge$
$((\mathrm{q}^i_{\mathrm{ontable}} = \mathrm{q}^{i+1}_{\mathrm{ontable}}) \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1})$
$\quad \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1})) \wedge$

on:

$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1} \wedge \overrightarrow{x_2^i} = \overrightarrow{y_2}) \implies \mathrm{q}^i_{\mathrm{on}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1} \wedge \overrightarrow{x_2^i} = \overrightarrow{y_2}) \implies \mathrm{q}^{i+1}_{\mathrm{on}} \wedge$
$(\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1} \wedge \overrightarrow{x_2^i} = \overrightarrow{y_2}) \implies \neg\, \mathrm{q}^{i+1}_{\mathrm{on}} \wedge$
$((\mathrm{q}^i_{\mathrm{on}} = \mathrm{q}^{i+1}_{\mathrm{on}}) \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{stack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1} \wedge \overrightarrow{x_2^i} = \overrightarrow{y_2})$
$\quad \vee (\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{unstack}) \wedge \overrightarrow{x_1^i} = \overrightarrow{y_1} \wedge \overrightarrow{x_2^i} = \overrightarrow{y_2}))$

Figure 1: Ungrounded Encoding for blocks-world for $k = 2$

The corresponding Ungrounded Encoding is:

$$\exists \mathrm{A}^0, \mathrm{PM}^0, \ldots, \mathrm{A}^{k-1}, \mathrm{PM}^{k-1} \forall \mathrm{OC} \exists \mathrm{P}^0, \ldots, \mathrm{P}^k$$
$$\mathrm{I_u}(\mathrm{P}^0, \mathrm{OC}) \wedge \mathrm{G_u}(\mathrm{P}^k, \mathrm{OC}) \wedge$$
$$\bigwedge_{i=0}^{k-1} \mathrm{T_u^i}(\mathrm{P}^i, \mathrm{P}^{i+1}, \mathrm{OC}, \mathrm{A}^i, \mathrm{PM}^i) \wedge \bigwedge_{i=0}^{k-1} \mathrm{RC}(\mathrm{A}^i, \mathrm{PM}^i)$$

In the sequel, we will define the auxiliary constraints for the initial and goal states and the transition functions. The domain restriction RC does not depend on the universal variables OC, so it remains unchanged from the SAT encoding (Definition 9). We refer to the example in Figure 1 for an illustration of the encoding running example of Listing 1. At the end of this section, we comment on the equivalence of the SAT and the QBF encoding. Basically, when the universal variables in the ungrounded QBF encoding are expanded, the resulting conjunction of constraints is equivalent to the constraints in the intermediate SAT encoding.

The initial and goal constraints must apply to particular object combinations, so for each predicate, we specify in which for-all branches, i.e., for which objects instantiations, they are positive or negative:

Definition 10 *The **Ungrounded Initial** constraint is defined as* $\mathrm{I_u}(\mathrm{P}^0, \mathrm{OC})$*:*

$$\bigwedge_{\mathrm{p} \in \mathrm{P}} \Big(\bigvee_{\mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{I}} \bigwedge_{j=1}^{|\overrightarrow{\mathrm{o}}|} (\mathrm{bin}(\mathrm{o}_j) = \overrightarrow{\mathrm{y}_j}) \Big) \iff \mathrm{q_p^0}$$

Definition 11 *The **Ungrounded goal** constraint is defined as* $\mathrm{G}_u(\mathrm{P}^k, \mathrm{OC}) = \mathrm{G}^+ \wedge \mathrm{G}^-$*, where*

$$\mathrm{G}^+ = \bigwedge_{\mathrm{p} \in \mathrm{P}} \Big(\bigvee_{\mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{g}^+} \bigwedge_{j=1}^{|\overrightarrow{\mathrm{o}}|} (\mathrm{bin}(\mathrm{o}_j) = \overrightarrow{\mathrm{y}_j}) \Big) \implies \mathrm{q_p^k},$$

$$\mathrm{G}^- = \bigwedge_{\mathrm{p} \in \mathrm{P}} \Big(\bigvee_{\mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{g}^-} \bigwedge_{j=1}^{|\overrightarrow{\mathrm{o}}|} (\mathrm{bin}(\mathrm{o}_j) = \overrightarrow{\mathrm{y}_j}) \Big) \implies \neg\, \mathrm{q_p^k}$$

For each predicate, if any of the action constraints is true in the preconditions or effects, then the corresponding variable is constrained to the appropriate value. We will expand on the action constraints $\mathrm{AC_p^u}$ in the definition 13.

Definition 12 *The **ungrounded transition function***

$$\mathrm{T_u^i}(\mathrm{P}^i, \mathrm{P}^{i+1}, \mathrm{OC}, \mathrm{A}^i, \mathrm{PM}^i) = \bigwedge_{\mathrm{p} \in \mathrm{P}} \mathrm{UPC_p^i}$$

*where the **ungrounded predicate constraint*** $\mathrm{UPC_p^i}$ =

$$\begin{aligned}
&(\mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{pre}^+) \implies \mathrm{q_p^i}) \wedge \\
&(\mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{pre}^-) \implies \neg\, \mathrm{q_p^i}) \wedge \\
&(\mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{eff}^+) \implies \mathrm{q_p^{i+1}}) \wedge \\
&(\mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{eff}^-) \implies \neg\, \mathrm{q_p^{i+1}}) \wedge \\
&((\mathrm{q_p^i} = \mathrm{q_p^{i+1}}) \vee \mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{eff}^+) \vee \\
&\quad \mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \mathrm{eff}^-))
\end{aligned}$$

The constraints are similar to the SAT encoding. For example, the positive precondition constraint states that a predicate must have been true at step i, whenever some action with this predicate in its precondition and matching action parameters occurs at time step i. The final constraint is called the frame axiom; it says that either the value of predicate stays the same, or some matching positive or negative effect has occurred. Next, we define in which OC-branch action A^i with parameters PM^i matches condition/effect Φ.

Definition 13 *Given* $\Phi \in \{\mathrm{pre}^+, \mathrm{pre}^-, \mathrm{eff}^+, \mathrm{eff}^-\}$*, the **ungrounded action constraint***

$$\mathrm{AC_p^u}(\mathrm{A}^i, \mathrm{PM}^i, \mathrm{OC}, \Phi) = \bigvee_{\mathrm{a} \in \mathrm{A}} \big((\overrightarrow{\mathrm{A}^i} = \mathrm{bin}(\mathrm{a})) \wedge \bigvee_{\mathrm{p}(\mathrm{x}_{l_1}, \ldots, \mathrm{x}_{l_n}) \in \Phi(\mathrm{a})} \bigwedge_{j=1}^{n} (\overrightarrow{\mathrm{x}_{l_j}^i} = \overrightarrow{\mathrm{y}_j}) \big)$$

Similar to the initial and goal constraints, the value of the predicate variables must be constrained in each for-all branch. As we discussed before, each for-all branch corresponds to one instantiation of object combinations. The ungrounded action constraint specifies which branches are evaluated to true. In plain words, for every for-all branch where the predicate parameters variables are equal to the universal variables, the ungrounded action constraint is evaluated to true. Note that the predicate parameters from the preconditions and effects are simply a subset of the action parameter variables, by the static semantics of PDDL.

Regarding the frame axioms, we use the following performance optimizations in our implementation: We do not propagate static predicates, i.e., predicates that do not appear in any of the action effects. Instead, we introduce only one instance, reused across all time steps. In particular, the type predicates in typed planning domains are handled as static predicates. Equality-predicates are treated specially, since they impose constraints between action parameters only, independent of the universal variables. We simply generate (in)equality constraints between action parameters directly, similar to the RC-constraint. We discuss this in more detail in the technical report (Shaik and van de Pol 2021).

Correctness

We demonstrate correctness, by showing how the SAT encoding can be stepwise transformed into the equivalent QBF encoding. Let $\delta = 2^{\gamma \times \eta}$ be the number of object combinations, and $n = |\,\mathrm{P}\,|$. Then the constraints on the propositions in time step i and $i+1$ in the SAT encoding are:

$$\exists \mathrm{f}^i_{\mathrm{p}_1(\overrightarrow{\mathrm{o}_1})} \ldots \exists \mathrm{f}^i_{\mathrm{p}_1(\overrightarrow{\mathrm{o}_\delta})} \quad \cdots \quad \exists \mathrm{f}^i_{\mathrm{p}_n(\overrightarrow{\mathrm{o}_1})} \ldots \exists \mathrm{f}^i_{\mathrm{p}_n(\overrightarrow{\mathrm{o}_\delta})} \bigwedge_{\mathrm{p} \in \mathrm{P}} \bigwedge_{\mathrm{p}(\overrightarrow{\mathrm{o}}) \in \mathrm{F}} \mathrm{PC}^i_{\mathrm{p}(\overrightarrow{\mathrm{o}})}$$

Here the existential blocks of propositional variables are grouped based on the predicate symbols. Since each of the propositions appears in only one constraint, we can push the

existential variables inside the conjunction. Thus the constraints are equivalent to the following groups of constraints in non-prenex form:

$$\exists \mathrm{f}^i_{\mathrm{p}_1(\overrightarrow{\mathrm{o}_1})} \cdots \exists \mathrm{f}^i_{\mathrm{p}_1(\overrightarrow{\mathrm{o}_\delta})} \bigwedge_{\mathrm{p}_1(\overrightarrow{\sigma})\in \mathrm{F}} \mathrm{PC}^i_{\mathrm{p}_1(\overrightarrow{\sigma})}$$
$$\wedge \cdots \wedge$$
$$\exists \mathrm{f}^i_{\mathrm{p}_n(\overrightarrow{\mathrm{o}_1})} \cdots \exists \mathrm{f}^i_{\mathrm{p}_n(\overrightarrow{\mathrm{o}_\delta})} \bigwedge_{\mathrm{p}_n(\overrightarrow{\sigma})\in \mathrm{F}} \mathrm{PC}^i_{\mathrm{p}_n(\overrightarrow{\sigma})}$$

Since all the constraints have a uniform shape, we can abbreviate the conjunction by a for-all quantification in QBF. So the previous formula is equivalent to the grouping with object combination variables in the QBF encoding:

$$\forall \mathrm{OC}\ (\exists \mathrm{q}^i_{\mathrm{p}_1}\ \mathrm{UPC}^i_{\mathrm{p}_1} \wedge \cdots \wedge \exists \mathrm{q}^i_{\mathrm{p}_n}\ \mathrm{UPC}^i_{\mathrm{p}_n})$$

Remember that the action and parameter variables are the same for both SAT and QBF encoding. This proves the equivalence between the SAT and the QBF encodings.

Implementation and Experiments

We implemented our encoding of PDDL problems to QBF problems in QCIR format as a Python program, which is available online.[2] The tool handles domains with types and equality predicates by treating them as static predicates; it also handles negative preconditions and constants, but it cannot yet handle domains with conditional effects. To solve the encoded problems for a given length k, we transform them to QDIMACS format, and use the QBF solver CAQE (Rabe and Tentrup 2015) with the internal preprocessor Bloqqer (Biere, Lonsing, and Seidl 2011). Some initial experiments with other preprocessors and solvers indicated that CAQE+Bloqqer is a very good combination; we leave a systematic comparison as future work. Finally, our tool extracts a concrete plan from CAQE's output, and validates that the plan is indeed correct. In the sequel, we refer to the combination of our encoding, CAQE and Bloqqer as "Q-Planner".

We performed two experiments: in the first experiment, we run Q-Planner on a number of Hard-to-Ground (htg) domains. These include planning problems from 4 Organic Synthesis domains: 2 from IPC-18 (non-split Satisfying and Optimizing track) and the 2 original benchmarks (Alkene and Mitexams) (Masoumi, Antoniazzi, and Soutchanski 2015). The simplified domains submitted to IPC-18 received an outstanding domain submission award. In addition, we also consider other Hard-to-ground domains (htg): Genome-edit-distance (without costs), Pipesworld from Corrêa et al. (2021) and Childsnack, Visitall, Blocks and Logistics from Lauer et al. (2021). For these domains, we consider a subset of instances which capture the hardness in grounding. We compare Q-Planner with 3 state-of-the-art planners, specified below.

In the second experiment, we encoded all problems from previous IPC planning competitions that our translation can handle, resulting in 18 domains. We are mainly interested in the number of instances that can be solved in each domain (within a given time and memory limit). In this experiment, we compared Q-Planner with Madagascar, using a simple SAT encoding (no invariants or parallel plans) to compare the effect of our ungrounded encoding on the encoding size, solving time, and memory usage.

[2]https://github.com/irfansha/Q-Planner

		QBF/SAT		Non-SAT	
	Domains (Total)	QP	M	FDS	PL
LA	Alkene (18)	18	1	18	18
	OS-Sat18 (20)	12	-	3	15
	OS-Opt18 (20)	18	-	9	20
	MitExams (20)	6	-	1	12
	Ged (20)	7	5	20	20
	Pipesworld (20)	16	9	19	18
LP	Childsnack (16)	6	11	16	15
	Visitall (18)	2	-	6	18
LO	Blocks (12)	6	4	12	11
	Logistics (12)	-	-	12	12
	Total (176)	91	30	116	159

Table 1: Instances solved with 300 GB memory and 3 hour time limit. Running 4 tools on 10 Hard-to-ground domains, among them 6 large action arity (LA), 2 large predicate arity (LP), and 2 large objects (LO) domains.

Experimental Setup

For the first experiment, we compare Q-planner (QP) with 3 other planners: (1) Madagascar (version M) (Rintanen 2014) with relaxed existential step encoding and invariant synthesis, a SAT based planner based on grounding; (2) Fast Downward Stone Soup 2018 (FDS) (Helmert and Röger 2011), a non-SAT based planner which was the winner of IPC-2018 competitions in the satisfying track; and (3) Powerlifted (PL) (Corrêa et al. 2020), a non-SAT based planner which avoids grounding and is the state-of-the-art for Organic Synthesis. We use recommended configurations for all three planners for fair comparisons. We allow 300GB main memory and 3 hours per instance.

For the second experiment, we want to compare the ungrounded QBF encoding (using Q-Planner) with the grounded SAT encoding (using Madagascar). To eliminate other factors, we use Madagascar in its simplest configuration without invariants and without parallel plans (here called M-simple), i.e., a standard sequential SAT encoding with direct encoding of objects and actions. Since the CAQE solver calls SAT-solver Crypto-minisat (Soos, Nohl, and Castelluccia 2009), we also use Crypto-minisat to solve the SAT encodings in Madagascar. We allow 8GB main memory and 5000 seconds time for solving each instance.

For both experiments, we increase the path length in steps of 5 (consistent with Madagascar). We ran all computations for our experiments on the Grendel cluster.[3]

[3]http://www.cscaa.dk/grendel/, each problem uses one core on a Huawei FusionServer Pro V1288H V5 server, with 384 GB main memory and 48 cores of 3.0 GHz (Intel Xeon Gold 6248R).

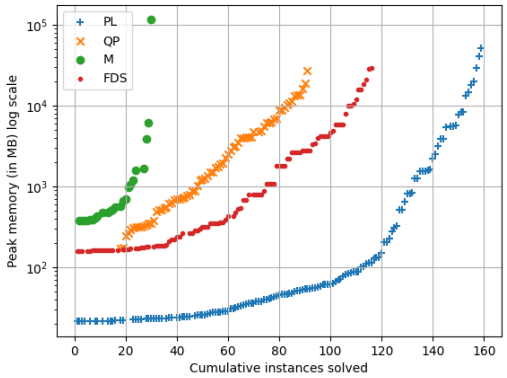

Figure 2: htg instances solved in given peak memory.

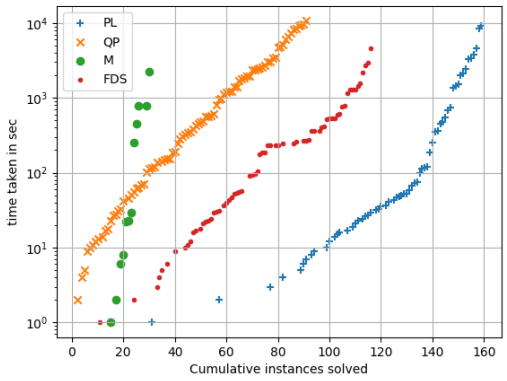

Figure 3: htg instances solved in given time.

Results

We now discuss the results of the experiments.[4] For the first experiment (Table 1), Q-Planner solves 54 Organic Synthesis instances where as Madagascar only solves 1 using 115GB memory. These results are consistent with the IPC-2018 competitions, where 2 variants of Madagascar could not solve any of the (non-split) organic synthesis benchmarks in the satisfying track based on SAT solving. This is due to the presence of actions with many parameters (up to 31), thus even with action splitting and other optimizations the grounded encodings are too big. In total, Q-Planner solves 91 instances whereas Madagascar only solves 30 instances. For the Childsnacks domain Madagascar performs better than Q-Planner. This happens only for instances with a few action parameters, where Madagascar finds long plans quickly, using parallel plans. In most cases, Madagascar exhausts all 300GB memory before reaching the time limit, highlighting the memory bottleneck in htg domains.

We also provide cactus plots showing how many instances can be solved within a given memory (Fig. 2) or time limit (Fig. 3). In both aspects, Q-Planner clearly outperforms Madagascar significantly in the htg domains.

[4]All benchmarks, logs and statistics are avaialable at Zenodo, https://doi.org/10.5281/zenodo.6367523.

	Domains	Q-planner #solved	Q-planner size	M-simple #solved	M-simple size
Typed IPC domains	Blocks	28	0.14	**30**	4.8
	DriverLog	**11**	0.27	**11**	5.7
	Elevator	34	0.20	**43**	2.1
	Hiking	**9**	0.49	6	75.8
	Satellite	**6**	0.23	**6**	1.7
	Tidybot	1	1.7	**13**	519.9
	Termes	2	0.68	**3**	14.3
	Thoughtful	0	1.7	**5**	857.5
	Visitall	7	0.41	**13**	2.5
Untyped IPC domains	Blocks-3op	17	0.16	**19**	63.2
	Movie	**30**	0.15	**30**	0.28
	Depot	4	0.32	**6**	70.9
	Gripper	3	0.19	**5**	0.80
	Logistics	**12**	0.34	**12**	4.0
	Mprime	17	0.36	**34**	1954
	Mystery	3	0.36	**13**	399.1
	Grid	**2**	0.52	**2**	47.9
	Freecell	2	1.1	**6**	631.1
	Total sum	188	9.32	257	4656

Table 2: Instances solved within 5000s and 8GB internal memory. We also show the size (in MB) of the encoding for the maximum instance after preprocessing (if possible).

Concerning the non-SAT based planners: FDS performs well, solving 116 instances. On most of the remaining instances it exhausts all memory in the grounding phase. The Powerlifted planner solves 159 instances. Its excellent performance is as expected, since it also avoids grounding on actions while using optimized database queries on grounded states. Still, there are some Organic Synthesis instances where Powerlifted exhausts all memory. Q-Planner, on the other hand, might exceed the time-limit, but never used more than 28GB, even for refuting the existence of a plan of shorter length, when it could not solve the instance completely. The technical report contains more experiments and information on optimal plans (Shaik and van de Pol 2021). Interestingly, Q-Planner solves one instance uniquely in the hardest set of benchmarks, i.e., MitExams.

Concerning the results of the second experiment (Table 2), the Ungrounded QBF Encoding never exceeds 1.7MB, so it is orders of magnitude more compact than the simple SAT encoding, which sometimes exceeds 1950MB. When comparing the performance of solving, the situation is different: In most domains, Madagascar solves more instances than Q-Planner, using less time and memory, despite the larger encodings. This is consistent with the observations of Cashmore, Fox, and Giunchiglia (2012) on SAT vs QBF.

From the results on 18 IPC domains, it is clear that (at least with the current QBF solvers) our QBF encoding is not a replacement for SAT encodings, but rather a complement for domains where grounding is hard.

Related Work

We first discuss some approaches in the literature that avoid grounding in SAT encodings by allowing quantification, using EPR, CP, or SMT. Then we discuss related work on QBF.

Navarro Pérez and Voronkov (2013) presented an encoding of planning problems in EPR (Effectively Propositional Logic). While EPR can encode conditional effects, satisfiability is NEXPTIME-complete (Lewis 1980). Our QBF encoding stays in the 3rd level of the polynomial hierarchy, combining a compact encoding with more efficient solving.

Other approaches, such as numerical planning as SMT (Satisfiability Modulo Theories) by Bofill, Espasa, and Villaret (2016) and numerical planning as CP (Constraint Programming) by Espasa et al. (2019), also avoid grounding for compact encodings. It would be interesting to apply those approaches to Organic Synthesis.

Cashmore, Fox, and Giunchiglia (2013) proposed a QBF encoding with *partial grounding*, which uses universal variables for a subset of ungrounded propositions only. They use *universal variables for only one object*. This leads to problems with propagating untouched propositions. To avoid incorrect plans, they need to introduce the restriction that predicates can have at most one ungrounded parameter. This is costly: if a predicate parameter is grounded, several action parameters must also be grounded.

We see our work as a generalization of their technique. We use *object combinations* in the universal layer of the encoding, thus avoiding grounding completely. This results in more compact QBF encodings. The software for partial grounding encoding is not available for comparison. Partial grounding of some parameters is trivial in our approach (it can be done on the domain file directly). However, we expect that partial grounding would be suboptimal. In case of organic synthesis, partially grounding some predicates still results in very large encodings.

The main focus of compact QBF encodings for reachability, synthesis and planning has been on reducing the number of transition function copies (Rintanen 2001; Cashmore, Fox, and Giunchiglia 2012). While avoiding unrolling reduces the encoding size, it comes at the cost of losing inference between different time steps. On the other hand, our approach using universal quantification to avoid grounding, while keeping the inference between time steps, might be a better way to use the advances in the QBF solving for planning problems. Since the encodings are generated directly from (restricted) First-order Logic representations, the resulting encodings are compact and more scalable.

Conclusion and Future Work

In this paper, we propose the Ungrounded QBF Encoding for classical planning problems, which results in compact encodings by avoiding grounding completely. The size of this QBF encoding grows linearly in the number of action names, predicates and the path length, and logarithmic in the number of objects. We provide an open source implementation of the Ungrounded Encoding, translating classical planning problems in PDDL to QBF problems in QCIR format. The resulting QBF formulas can be preprocessed and solved by existing QBF tools (we used Bloqqer and CAQE).

The experiments show that our encoding effectively avoids the memory bottleneck due to grounding. This is relevant for domains with actions that involve many objects, such as Organic Synthesis. Our main result is to solve many instances of Organic Synthesis, which so far could not be handled by any SAT/QBF based planner. We solve 54 problems in the 4 Organic Synthesis domains. We compare our implementation with the state-of-the-art SAT based planner Madagascar, which only solves 1 instance, using 115GB memory. Even on other htg domains, we outperform Madagascar when grounding is the bottleneck. In total we solve 91 instances where as the Madagascar solves 30 instances. We also compare with the non-SAT based planners Fast Downward Stone Soup 2018 (FDS) and Powerlifted, which perform better than any SAT/QBF approach.

Future Work. There are several research directions from here. One direction is to extend the Ungrounded Encoding to planning problems with uncertainty or planning with adversaries. QBF encodings have been demonstrated to be useful for such domains, such as conformant planning (Rintanen 1999, 2007). To this end, the Ungrounded Encoding must be extended, in order to handle some dependencies between state predicates, and to handle conditional effects.

Another direction is to improve the efficiency of the Ungrounded Encoding: One could consider to incorporate automatically generated grounded invariants (Rintanen 2008) or lifted invariants (Fiser 2020) to speed up the search. Similarly, one could consider extending the Ungrounded Encoding with compact QBF encodings, such as the non-copying Iterative Squaring Encoding (Rintanen 2001) and the Compact Tree Encoding (Cashmore, Fox, and Giunchiglia 2012). These encodings could in principle lead to an even more concise encoding (logarithmic in the length of the plan), but we expect less spectacular benefits than compared to SAT encodings, since our encoding is already quite small. It would also be interesting to investigate ungrounded versions of *parallel plans* (Rintanen, Heljanko, and Niemelä 2006), but currently the sequentiality of plans is deeply embedded in our encoding.

Finally, we hope to inspire new research on preprocessing and solving QBF problems, by submitting our problems to the QBFeval competition. Our encodings are very concise, but the current QBF solvers do not yet take advantage of the structure of the Ungrounded Encoding. While SAT solvers are rather mature, the field of QBF solvers is still improving rapidly. An advance in QBF solvers could make the Q-Planner approach competitive to the SAT approach for general classical planning problems.

Acknowledgments

We thank Michael Cashmore for discussions on the partially grounded QBF encoding. We thank Leander Tentrup for support with the CAQE QBF solver, and Mikhail Soutchanski for the original Organic Synthesis benchmarks. The experimental results were obtained at CSCAA, the Centre for Scientific Computing, Aarhus.

References

Biere, A.; Lonsing, F.; and Seidl, M. 2011. Blocked Clause Elimination for QBF. In *Proceedings of Twenty-Third CADE*, volume 6803 of *Lecture Notes in Computer Science*, 101–115. Springer.

Bofill, M.; Espasa, J.; and Villaret, M. 2016. The RANTANPLAN planner: system description. *Knowl. Eng. Rev.*, 31(5): 452–464.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2012. Planning as Quantified Boolean Formula. In *Proceedings of Twentieth ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 217–222. IOS Press.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2013. Partially Grounded Planning as Quantified Boolean Formula. In *Proceedings of the Twenty-Third ICAPS*. AAAI.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the Thirty-First ICAPS*, 94–102. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proceedings of the Thirtieth ICAPS*, 80–89. AAAI Press.

Dershowitz, N.; Hanna, Z.; and Katz, J. 2005. Bounded Model Checking with QBF. In *Proceedings of the Eighth International Conference SAT*, volume 3569 of *Lecture Notes in Computer Science*, 408–414. Springer.

Espasa, J.; Coll, J.; Miguel, I.; and Villaret, M. 2019. Towards lifted encodings for numeric planning in Essence Prime. In *CP 2019 Workshop on Constraint Modelling and Reformulation*.

Fiser, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In *Proceedings of the Thirty-Fourth AAAI Conference*, 9835–9842. AAAI Press.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Haas, A. R. 1987. The case for domain-specific frame axioms. In *The Frame Problem in Artificial Intelligence*, 343–348. Elsevier.

Helmert, M.; and Röger, G. 2011. Fast Downward Stone Soup : A Baseline for Building Planner Portfolios. In *Seventh International Planning Competition*.

Jussila, T.; and Biere, A. 2007. Compressing BMC Encodings with QBF. *Electronic Notes in Theoretical Computer Science*, 174(3): 45–56.

Kautz, H. A.; and Selman, B. 1992. Planning as Satisfiability. In *Proceedings of the Tenth ECAI*, volume 92, 359–363.

Lauer, P.; Torralba, Á.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In Zhou, Z., ed., *Proceedings of the Thirtieth IJCAI*, 4119–4126. ijcai.org.

Lewis, H. R. 1980. Complexity results for classes of quantificational formulas. *Journal of Computer and System Science*, 21(3).

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling Organic Chemistry and Planning Organic Synthesis. In *GCAI*, volume 36 of *EPiC Series in Computing*, 176–195. EasyChair. Benchmarks available https://www.cs.ryerson.ca/mes/publications/.

Matloob, R.; and Soutchanski, M. 2016. Exploring Organic Synthesis with State-ofthe-Art Planning Techniques. In *ICAPS Workshop on Scheduling and Planning Applications*.

Navarro Pérez, J. A.; and Voronkov, A. 2013. Planning with Effectively Propositional Logic. In Voronkov, A.; and Weidenbach, C., eds., *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Computer Science*, 302–316. Springer.

Rabe, M. N.; and Tentrup, L. 2015. CAQE: A Certifying QBF Solver. In Kaivola, R.; and Wahl, T., eds., *FMCAD*, 136–143. IEEE.

Rintanen, J. 1999. Constructing Conditional Plans by a Theorem-Prover. *J. Artif. Intell. Res.*, 10: 323–352.

Rintanen, J. 2001. Partial Implicit Unfolding in the Davis-Putnam Procedure for Quantified Boolean Formulae. In *Proceeding of the Eighth International Conference LPAR*, volume 2250 of *Lecture Notes in Computer Science*, 362–376. Springer.

Rintanen, J. 2007. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the Twenty-Second AAAI Conference*, 1045–1050.

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In *Proceeding of the Eighteenth ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, 568–572. IOS Press.

Rintanen, J. 2014. Madagascar: Scalable Planning with SAT. In *Proceeding of the Eighth IPC*.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12-13): 1031–1080.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2009. SAT-Based Parallel Planning Using a Split Representation of Actions. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth ICAPS*. AAAI.

Shaik, I.; and van de Pol, J. 2021. Classical Planning as QBF without Grounding (extended version). *ArXiv/CoRR*, 2106.10138.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *Proceedings of the Twelfth International Conference SAT*, volume 5584 of *Lecture Notes in Computer Science*, 244–257. Springer.