Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Partially Grounded Planning as Quantified Boolean Formula

Michael Cashmore
University of Strathclyde
Glasgow, G1 1XH, UK
michael.cashmore@strath.ac.uk

Maria Fox
King's College London
London WC2R 2LS
maria.fox@kcl.ac.uk

Enrico Giunchiglia
Universit di Genova
16145 Genova (GE), Italy
giunchiglia@unige.it

Abstract

This paper describes a technique for translating bounded propositional reachability problems, such as Planning, into Quantified Boolean Formulae (QBF). The key feature of this translation is that the problem, and the resultant encoding is only partially grounded. The technique is applicable to other SAT or QBF encodings as an additional improvement, potentially reducing the size of the resulting formula by an exponential amount. We present experimental results showing that the approach applied to a simple SAT translation greatly improves the time taken to encode and solve problems in which there are many objects of a single type, even solving some problems that cannot be reasonably encoded as SAT.

1 Introduction

Planning as Satisfiability is one of the most well-known and effective techniques for classical planning: SATPLAN (Kautz and Selman 1992; 1996) was an award-winning system in the deterministic track for optimal planners in the first International Planning Competition (IPC) in 1998, the 4th IPC in 2004, and the 5th IPC in 2006. The basic idea is to encode the existence of a plan with $n+1$ (or fewer) steps as a propositional satisfiability (SAT) formula obtained by unfolding, n times, the symbolic transition relation of the automaton described by the planning problem.

In general, SAT-based planning, though quite successful, suffers from the drawback that it is easy to come up with problems in which the number of steps required is large, making it impossible to even encode the original problem as a propositional formula. The same problem arises in bounded model checking (Biere et al. 1999). As a solution to this problem we introduce a technique for translating bounded propositional planning problems into Quantified Boolean Formulae (QBF) in which objects are described in terms of equivalence classes.

The use of compact encoding as Quantified Boolean Formulae (QBFs) combined with the use of QBF solvers has been proposed (Dershowitz, Hanna, and Katz 2005; Jussila and Biere 2007; Mangassarian, Veneris, and Benedetti 2010; Cashmore and Fox 2010) as an approach to Planning. In particular, Rintanen (2001), Jussila and Biere (2007) and Cashmore et al. (2010; 2012) present encodings that are logarithmic in the number of time-steps in the problem, resembling the proof of the PSPACE-hardness of solving QBFs (Savitch 1970; Stockmeyer and Meyer 1973). In contrast our approach is linear in the number of time-steps, more closely resembling existing SAT-based encodings. Our technique translates bounded propositional planning problems into Quantified Boolean Formulae (QBF) in which objects are described in terms of equivalence classes.

Before being encoded as SAT, Planning problems are typically grounded. There is growing interest in the Planning community (Nguyen and Kambhampati 2001; Ridder and Fox 2011) in the question of how to avoid grounding, and the work described here shows how grounding can be avoided. It is clear that there is the potential for lifted encodings to be exponentially smaller than grounded ones.

The Quantified Boolean Formulae obtained from our translation can require exponentially fewer variables (and clauses) than a corresponding SAT encoding to describe the state (and transition relation). The potentially exponential reduction is a function of the objects in the domain and is greatest in domains in which there are many objects of a single type. In order to determine the effectiveness of the encoding we run experiments on selected domains showing that the QBF encoding scales better as the number of objects, and size of the instance, increases. In addition we show that there exist problems which can be solved by the QBF translation that remain unsolved by SAT.

After some preliminaries we describe the partially grounded QBF translation in Section 3 This is followed by a full example in Section 4 and some experimentation in Section 5 Finally, we conclude in Section 6

2 Preliminaries

Quantified Boolean Formula

Quantified Boolean Formulae (QBF), which is PSPACE-complete (Savitch 1970; Stockmeyer and Meyer 1973), is perhaps the most fundamental problem in PSPACE. An instance of the QBF problem is typically presented as a Boolean expression in conjunctive normal form (CNF).

The decision problem is stated as: given φ, a Boolean expression in conjunctive normal form, with Boolean variables $x_1, \ldots, x_n$ partitioned into m sets $X_1, \ldots, X_m$, is it

true that there exists an assignment to variables X_1 such that for all assignments made to X_2 there exists an assignment to X_3 (and so on) such that φ is satisfied? For example:

$$\exists x_1, x_2, \forall x_3, \exists x_4 \cdot (\\ (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4))$$

is false, since there is no assignment to variables x_1, x_2 and x_4 that will satisfy the expression for both values of x_3.

The QBF instance can be thought of as a binary tree with conjunctive and disjunctive nodes. The outermost quantifier can be expanded, removing it from the problem. Where $\phi[x/\top]$ is the result of assigning $x \rightarrow true$ in ϕ (and similarly for $\phi[x/\bot]$) the formula after expanding the outermost variable x becomes $\phi[x/\top] \vee \phi[x/\bot]$ if x is quantified existentially and $\phi[x/\top] \wedge \phi[x/\bot]$ if x is quantified universally. After every variable is expanded the formula is reduced to $true$ or $false$.

Variables that are not of the outermost set can also be expanded, but the definition differs. Only variables quantified after the expanded variable are copied into a new formula, while the preceding variables are present in both halves of the expansion. When given the expression

$$\exists X_1, \forall y, \exists X_2, \ldots, \exists X_m(\varphi)$$

expanding y will produce the equivalent formula:

$$\exists X_1 \left(\begin{array}{c} \exists X_2, \ldots, \exists X_m(\varphi[y/\top]) \\ \wedge \\ \exists X'_2, \ldots, \exists X'_m(\varphi[y/\bot]) \end{array} \right)$$

Expanding every universally quantified variable in the problem will flatten the formula to an exponentially larger SAT instance.

The Planning Problem

Although the idea of partially grounding with QBF can be applied to any representation, Planning problems here are specified using the standard STRIPS formulation (Fikes and Nilsson 1971). The world is described with F, a set of *fluents*. An assignment of true or false to these fluents describes a state. I, a complete assignment to F, denotes the initial state of the world. The goal state is described by G, a formula over F. The world is changed using A, the set of *action fluents*.

Traditionally, in order to translate the planning problem into Boolean encodings we must first ground the instance. Grounding means generating fluents and action fluents from the operators and propositions of the domain. Examples of propositions and operators are shown in the example domain in Figure 1. Briefly, the set of fluents are found by making every possible valid binding of objects to propositions. Action fluents are similarly created from the operators.

The encoding described in Section 3 is only partially grounded. This means that some operators will not be grounded at all, or only some parameters will be bound, creating a set of partially grounded operators. These operators will be represented by variables in the QBF encoding.

Planning as SAT

Consider a planning problem $\Pi = \langle F, A, I, G \rangle$. As standard in planning as satisfiability, the existence of a parallel plan with makespan n is proved by building a propositional formula with n copies of the sets F and A.

In the following,

- by X_α we denote one such copy of the set of variables;
- by $I(X_\alpha)$ (resp. $G(X_\alpha)$) we denote the formula obtained from I (resp. G) by substituting each $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$;
- by $\sigma(X_\alpha)$ we denote the formula obtained from σ by substituting each variable $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$;
- by $\tau(X_\alpha, X_\beta)$ we denote the formula obtained from τ by substituting each variable $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$ and similarly each $x' \in X'$ with the corresponding $x_\beta \in X_\beta$.

The state constraint $\sigma(X)$ enforces that:

- each action fluent in X implies a conjunction of fluents (in X) corresponding to its preconditions; and
- each pair of action fluents in X that are *mutually exclusive* form a binary disjunction of their negations.

The transition relation $\tau(X, X')$ enforces that:

- each action fluent in X implies its effects in X'; and
- each fluent in X' implies a disjunction of supporting actions and itself in X, (explanatory frame axioms).

For $n \geq 1$, the *planning problem* Π *with makespan* n is the Boolean formula Π_n defined as

$$\begin{array}{l} I(X_1) \wedge \bigwedge_{i=1}^{n} \sigma(X_i) \\ \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge G(X_{n+1}) \end{array} \quad (1)$$

and a *plan for* Π_n is an interpretation satisfying (1).

However, the size of (1) can be exponential in the number of fluents - making it impossible to even build (1). QBFs are a promising alternative representation language given that:

1. there exist encodings of the planning problem with makespan n as QBFs which are polynomial in the number of fluents (Rintanen 2001; Cashmore and Fox 2010; Cashmore, Fox, and Giunchiglia 2012), and
2. there is a growing interest in developing efficient solvers for QBFs; see, for example, the report from the 2010 QBF competition (Peschiera et al. 2010).

3 Partially grounded QBF encoding

Briefly, the idea behind the encoding is to represent a single object of the ungrounded type and then to use universal variables to create multiple contexts for this representation. When the universal variables are expanded - as described in Section 2 - the existential variables representing the object are copied, and multiple objects of the same type can be represented.

The new encoding will be introduced in two parts. First the state representation will be described. This will be followed by an in-depth description of the state and transition

constraints. The pigeonhole problem will be used as an example to illustrate the encoding. This domain was chosen as it is very simple to understand and an obvious candidate for lifting. The domain is described in Figure 1.

Splitting propositions and operators

Kautz and Selman (1992) used the idea of operator splitting to significantly reduce the size of the resulting encoding. The basic idea is to reduce the arity of operators by replacing operators that take three or more parameters by several operators that take no more than two parameters. For example, a $place\,(?p - pigeon,\ ?h - pigeonhole)$ operator for placing a pigeon p into pigeonhole h would be replaced by two split operators: $place[1](?p)$ and $place[2](?h)$.

Operator splitting has been explored and implemented in more detail in more recent SAT-based planners (Ernst, Millstein, and Weld 1997; Robinson et al. 2008; 2009) and bears some similarity to the alternate state-representation used by Huang et al. (2012).

The encoding presented here uses a split representation similar to that of Kautz and Selman (1992) in a semi-parallel setting. An operator with multiple parameters will be split into a number of split operators equal to the number of parameters to remain ungrounded plus one. For example, if only pigeons were to remain ungrounded in the example, splitting the $place$ operator will result in the split operators $place[1](?p)$ and $place[2](?h)$. However, if both objects were to remain ungrounded then the operator is split into three parts: $place[0]$; $place[1](?p)$; and $place[2](?h)$. The purpose of this third split operator, $place[0]$, is to ensure consistency between the leaves of the QBF. This role will be explained in detail later.

The propositions are also split, however, no extra split proposition is added. Instead the proposition is split into a number of parts equal to the number of ungrounded parameters and each of these split propositions has an arity of the number of grounded parameters. For example, when both pigeons and pigeonholes remain ungrounded, the proposition $in(?p, ?h)$ becomes: $in[1](?p)$ and $in[2](?h)$.

Once this is combined with partial grounding, a proposition with arity 3 is described with only 3 variables, as opposed to grounding fully without splitting, in which case there are $3^{|O|}$ grounded action fluents. $|O|$ is the number of objects in the problem.

Partially grounded state representation

After splitting, the split propositions and operators that correspond to parameters which are not to be lifted are grounded. The resulting sets are encoded as sets of Boolean variables: grounded split fluents (F); grounded split action fluents (A); ungrounded split propositions (P); and ungrounded split operators (O). A variable from one of these sets is given a subscript to represent the parameter of which it is representative. For example $o_\alpha \in O$ is a split operator variable representing the parameter α. Each state is encoded as the set X, comprising of two parts: X^g and X^u, where $X^g := F \bigcup A$ and $X^u := P \bigcup O$.

Additional variables are required in order to ensure that the ungrounded parts of the plan are consistent between contexts of the QBF. These variables will be called *lock* variables. In each case, m is the number of universal variables. For each split operator variable o_α a set of lock variables are added to X^g:

$$\{lock_1\_o_\alpha, \ldots, lock_m\_o_\alpha\} \in X^g$$

The operator lock variables ensure that a variable such as $place[1](?p)$ can only be made true in one context of the QBF at each time-step. This is important as otherwise we could place all the pigeons into a single pigeonhole with a single action.

For each split proposition p_α a set of lock variables is added to X^u for each *other* ungrounded parameter of the proposition. For example, consider split proposition:

$$p_\alpha := in[1](?p)$$

A lock is added to X^u corresponding to the other ungrounded parameter $p_\beta := in[2](?h)$. These variables are denoted

$$\{lock_1\_p_\beta, \ldots, lock_m\_p_\beta\} \in X^u$$

Similarly for split proposition $p_\beta := in[1](?h)$ a set of lock variables are added:

$$\{lock_1\_p_\alpha, \ldots, lock_m\_p_\alpha\} \in X^u$$

The proposition lock variables ensure that the same object is bound to the proposition between time-steps. For example, if $in[1](?p)$ and $in[2](?h)$ were both true in two different contexts of the QBF, the proposition lock variables are required to know which pigeon is in which pigeonhole.

In the pigeonhole example, in which both objects are ungrounded, the set $X := X^g \bigcup X^u$ is

$$\begin{aligned}
X_i^g := \{ & \\
& place[0]_i, \\
& lock_1_place[1]_i, \ldots, lock_m_place[1]_i, \\
& lock_1_place[2]_i, \ldots, lock_m_place[2]_i\} \\
X_i^u := \{ & \\
& place[0]_i,\ place[1]_i, \\
& in[0]_i,\ in[1]_i, \\
& placed_i,\ empty_i, \\
& lock_1_in[0]_i, \ldots, lock_m_in[0]_i, \\
& lock_1_in[1]_i, \ldots, lock_m_in[1]_i\}
\end{aligned}$$

As the number of pigeons or pigeonholes is increased, the size of the lock variable sets grows logarithmically. No other variables are added.

A number of variables are quantified universally between X^g and X^u. An encoding of a planning problem with makespan n contains $n+1$ copies of X, and so the quantification layer is:

$$\exists X_1^g \ldots X_{n+1}^g \forall a_1 \ldots a_m \exists X_1^u \ldots X_{n+1}^u$$

The universal variables $a_1, \ldots, a_m$ define 2^m contexts - leaves of the QBF tree - each of which encodes a unique object of each ungrounded type.

The Partially Grounded QBF (PG-QBF) encoding of a planning problem with makespan n is the Quantified Boolean formula Φ_n containing $n+1$ copies of X and is defined by:

```
(define (domain PIGEONHOLE)
  (:requirements :strips :typing)
  (:types pigeon pigeonhole)
  (:predicates (in ?p - pigeon ?h - pigeonhole)
               (placed ?p - pigeon)
               (empty ?h - pigeonhole)
               )

  (:action place
             :parameters (?p - pigeon ?h - pigeonhole)
             :precondition (and (empty ?h)
                                (not (placed ?p)))
             :effect
             (and (and (not (empty ?h))
                  (in ?p ?h))
                  (placed ?p))))
```

Figure 1: The domain for the pigeonhole problem with operator $place$ and propositions $placed$, $empty$, and in.

$$\begin{aligned}&\exists X_1^g \ldots X_{n+1}^g \forall a_1 \ldots a_m \exists X_1^u \ldots X_{n+1}^u (\\ &\quad I(X_1^g \cup X_1^u) \wedge G(X_{n+1}^g \cup X_{n+1}^u) \\ &\quad \wedge \textstyle\bigwedge_{i=1}^{n} \tau_{qbf}(X_i^g \cup X_i^u, X_{i+1}^g \cup X_{i+1}^u) \\ &\quad \wedge \textstyle\bigwedge_{i=1}^{n} \sigma_{qbf}(X_i^g \cup X_i^u))\end{aligned} \tag{2}$$

A *plan for* Φ_n is an interpretation satisfying (2).

The state constraints $\sigma_{qbf}(X_i^g \cup X_i^u)$ ensure that $O_i \bigcup A_i$ represent a valid action choice, and that their preconditions hold in X_i. The transition constraints

$$\tau_{qbf}(X_i^g \cup X_i^u, X_{i+1}^g \cup X_{i+1}^u)$$

ensure that the effects of each action applied in step i hold in step $i+1$, and also enforce the frame axioms.

State constraints

The state constraints $\sigma_{qbf}(X_i^g \cup X_i^u)$ ensure that if an action is to be applied then:

1. exactly one split action variable is made true for each grounded parameter;
2. for each ungrounded parameter, the corresponding split operator variable is made true in exactly one context of the QBF;
3. any action with which it is mutually exclusive cannot be applied; and
4. it's preconditions hold.

In the following the time-step subscript i is omitted for simplicity.

Constraint (1) is defined by:

$$\begin{array}{ll} a_\alpha \rightarrow \neg b_\alpha & for\,all\,a_\alpha, b_\alpha \in A,\, a_\alpha \neq b_\alpha \\ o_\alpha \rightarrow (a_{\beta,1} \vee, \ldots, \vee a_{\beta,j}) & for\,all\,o_\alpha \in O\,and\,a_\beta \in A \end{array}$$

where $a_{\beta,i}$ represents the ith grounded split action fluent representing the parameter β. j is the number grounded split action fluent variables representing this action and parameter combination in A.

These constraints ensure that only a single split action fluent for each parameter is made true and that if a split operator variable is made true, representing an ungrounded part of the action, the grounded part must also be made true. For example, if pigeons are grounded, but pigeonholes remain ungrounded:

$$\begin{aligned} &place[1](pigeon_i) \rightarrow \neg place[1](pigeon_j), \\ &\quad for\,each\,i,j \in |P|,\, i \neq j \\ &place[2](?h) \rightarrow \\ &\quad place[1](pigeon_1) \vee \ldots \vee place[1](pigeon_{|P|}) \end{aligned}$$

In the case where both object types are ungrounded:

$$\begin{aligned} &(place[1](?p) \rightarrow place[0]) \wedge \\ &(place[2](?h) \rightarrow place[0]) \end{aligned}$$

Constraint (2) makes use of the lock variable set associated with the split operator:

$$\begin{array}{ll} o_\alpha \rightarrow (lock_j\_o_\alpha \leftrightarrow a_j), & for\,j = 1, \ldots, m \\ a_\beta \wedge \bigwedge_{j=1}^{m} (lock_j\_o_\alpha \leftrightarrow a_j) \rightarrow o_\alpha & \end{array}$$

The first constraint ensures that the split operator variable o_α is true in at most one context of the QBF. The second constraint ensures that if a grounded split action fluent is made true then the split operator variable o_α is true in exactly one context of the QBF. For example, consider

$$place[1](?p) \rightarrow (lock_1\_place[1] \leftrightarrow a_1)$$

in an encoding with m=1.

If $place[1](?p) \models \top$ in the context defined by

$$a_1 \models \bot$$

then the associated lock variable $lock_1\_place[1] \models \bot$.

Now, when $a \models \top$, $place[1](?p)$ cannot be true, as this implies $lock_1\_place[1]$ must be true, which causes a conflict. The key is that $lock_1\_place[1]$ is quantified *before* the universal variables, and so is not copied upon expansion.

The second constraint:

$$\begin{aligned} &place[0] \wedge (lock_j\_place[1] \leftrightarrow a_j) \rightarrow place[1](?p) \\ &\wedge \\ &place[0] \wedge (lock_j\_place[2] \leftrightarrow a_j) \rightarrow place[2](?h) \end{aligned}$$

simply ensures that both halves of $place(?p, ?h)$ are performed. If a grounded part of the action is true, then all of the split operators corresponding to ungrounded parameters of the action must also be performed. Otherwise only part of an action is performed. It is for this reason that the additional split operator ($place[0]$) is created.

The use of operator lock variables to enforce disjunction between instances of the same variable in different contexts of the QBF is the most important contribution of this encoding.

Constraint 3 is easily enforced in exactly the same way as a SAT encoding, with binary disjunctions between the negations of grounded split action variables.

Constraint 4 is enforced in two parts. Firstly:

$$o_\alpha \rightarrow p_\alpha$$
$$a_\alpha \rightarrow f_\alpha$$

for each $o_\alpha \in O$ with associated split proposition precondition p_α, and each split action fluent $a_\alpha \in A$ with associated split fluent precondition f_α. For example:

$$place[1](?p) \rightarrow \neg placed(?p)$$
$$\wedge$$
$$place[2](?h) \rightarrow empty(?h)$$

Secondly a constraint is required to ensure that the split propositions and split fluents constituting the preconditions belong to the same fluent.

For example, consider the new action $remove(?p, ?h)$ in Figure 2. It is not enough to ensure that $in[0](?p)$ and $in[1](?h)$ are true in the correct contexts. It must also be ensured that both halves connect in the same whole fluent. Otherwise, if $pigeon1$ was in $pigeonhole1$ and $pigeon2$ in $pigeonhole2$ it would be possible to remove $pigeon1$ from $pigeonhole2$. This is avoided by adding the constraint:

$$o_\alpha \rightarrow \bigwedge_{j=1}^{m}(lock_j\_o_\beta \leftrightarrow lock_j\_p_\beta)$$

for each split operator o_α with precondition p_α that forms part of a whole fluent, and for each other ungrounded split proposition p_β of that fluent.

Using $remove$ as an example: the first ungrounded split operator variable $remove[1](?p)$ implies that the pigeon is in a pigeonhole ($in[1](?p)$) and also that the $in[1](?p)$ split proposition is related to the correct pigeonhole.

The pigeonhole object is remembered by the proposition lock variables $lock_1\_in[2], \ldots, lock_m\_in[2]$. The correct pigeonhole means the pigeonhole from which it is being removed, as stored in the operator lock variables $lock_1\_remove[2], \ldots, lock_m\_remove[2]$.

The resulting constraints are:

$$remove[1](?p) \rightarrow$$
$$\bigwedge_{j=1}^{m}(lock_j\_remove[2] \leftrightarrow lock_j\_in[2])$$
$$\wedge$$
$$remove[2](?h) \rightarrow$$
$$\bigwedge_{j=1}^{m}(lock_j\_remove[1] \leftrightarrow lock_j\_in[1])$$

and ensure that the pigeon is removed only from its own pigeonhole.

Transition constraints

The transition constraints $\tau_{qbf}(X_i, X_{i+1})$ ensure that:

1. X_{i+1} models the effects of the actions applied in X_i;
2. facts true in X_{i+1} and not added by an action in X_i were true in X_i;

Consider $\tau(X_i, X_{i+1})$; in the following description of the constraints we will use v_i and v_{i+1} to distinguish between variables belonging to the two sets, where $v_i \in X_i$ and $v_{i+1} \in X_{i+1}$.

Constraint (1) is enforced in much the same way as constraint (4) of the state constraints.

$$o_{\alpha,i} \rightarrow p_{\alpha,i+1}$$
$$a_{\alpha,i} \rightarrow f_{\alpha,i+1}$$

for each $o_\alpha \in O$ with associated split proposition effect p_α, and each split action fluent $a_\alpha \in A$ with associated split fluent effect f_α. Note that the effects may be negations. For example:

$$place[1]_i \rightarrow placed_{i+1}$$
$$\wedge$$
$$place[2]_i \rightarrow \neg empty_{i+1}$$

Additionally, the proposition lock variables must be set:

$$o_{\alpha,i} \rightarrow \bigwedge_{j=1}^{m}(lock_j\_o_{\beta,i} \leftrightarrow lock_j\_p_{\beta,i+1})$$

for each split operator o_α with effect p_α that forms part of a whole fluent, and for each other ungrounded split proposition p_β of that fluent. For example:

$$place[1]_i \rightarrow \bigwedge_{j=1}^{m}(lock_j\_place[2]_i \leftrightarrow lock_j\_in[2]_{i+1})$$
$$\wedge$$
$$place[2]_i \rightarrow \bigwedge_{j=1}^{m}(lock_j\_place[1]_i \leftrightarrow lock_j\_in[1]_{i+1})$$

Constraint 2 enforces the frame axioms. These enforce the requirement that split fluents and split propositions retain the correct value between states, and also that the locks are maintained. The first part of this is expressed with:

$$p_{\alpha,i} \rightarrow p_{\alpha,i+1} \vee \bigvee D_{p_{\alpha,i}}$$
$$f_{\alpha,i} \rightarrow f_{\alpha,i+1} \vee \bigvee D_{f_{\alpha,i}}$$
$$\neg p_{\alpha,i} \rightarrow \neg p_{\alpha,i+1} \vee \bigvee A_{p_{\alpha,i}}$$
$$\neg f_{\alpha,i} \rightarrow \neg f_{\alpha,i+1} \vee \bigvee A_{f_{\alpha,i}}$$

for each $p \in P$ and each $f \in F$. D_{p_α} is the set of split operators o_α that include p_α as a delete effect. A_{p_α} is the set of split operators o_α that include p_α as an add effect. D_{f_α} and A_{f_α} are similarly defined sets of action fluents.

In the example:

$$empty_i \rightarrow empty_{i+1} \vee place[2]_i$$
$$\wedge$$
$$placed_i \rightarrow placed_{i+1}$$
$$\wedge$$
$$\neg empty_i \rightarrow \neg empty_{i+1}$$
$$\wedge$$
$$\neg placed_i \rightarrow \neg placed_{i+1} \vee place[1]_i$$

The locks are maintained using the constraints:

$$p_{\alpha,i+1} \rightarrow \bigvee A_{p_{\alpha,i}} \vee \bigwedge_{j=1}^{m}(lock_j\_p_{\beta,i} \leftrightarrow lock_j\_p_{\beta,i+1})$$

```
(:action remove
          :parameters (?p - pigeon ?h - pigeonhole)
          :precondition (in ?p ?h)
          :effect
          (and (not (in ?p ?h)
             (and (empty ?h))
                (not (placed ?p))))))
```

Figure 2: The operator *remove* for the pigeonhole domain.

for each $p_\alpha \in P$ and each other parameter β of the whole fluent. For example:

$$in[1]_{i+1} \rightarrow \\ \quad place[1]_i \vee \bigwedge_{j=1}^{m}(lock_j\_in[2]_i \leftrightarrow lock_j\_in[2]_{i+1}) \\ \wedge \\ in[2]_{i+1} \rightarrow \\ \quad place[2]_i \vee \bigwedge_{j=1}^{m}(lock_j\_in[1]_i \leftrightarrow lock_j\_in[1]_{i+1})$$

These constraints ensure that the split proposition locks refer to the context in which the linked split proposition resides.

4 Example

Putting everything together we arrive at the QBF instance Φ_n, constructed according to formula 2, with the quantification layer:

$$\exists X_1^g \dots X_{n+1}^g \forall a_1 \dots a_m \exists X_1^u \dots X_{n+1}^u$$

where

$$X_i^g := \{ \\ \quad place[0]_i, \\ \quad lock_1\_place[1]_i, \dots, lock_m\_place[1]_i, \ \} \\ \quad lock_1\_place[2]_i, \dots, lock_m\_place[2]_i\} \\ X_i^u := \{ \\ \quad place[0]_i, place[1]_i, \\ \quad in[0]_i, in[1]_i, \\ \quad placed_i, empty_i, \\ \quad lock_1\_in[0]_i, \dots, lock_m\_in[0]_i, \\ \quad lock_1\_in[1]_i, \dots, lock_m\_in[1]_i\}$$

and the constraints are defined by Figure 3. Constraints 1 and 2 represent the initial and goal states respectively.

Constraints 3 to 8 ensure that only a single place action is attempted at each time-step, and that each split operator variable representing this action is true in only one context of the QBF.

Constraints 9 and 10 enforce action preconditions, while constraints 11 to 14 enforce the effects of these actions.

Constraints 15 to 24 are the frame axioms. Constraints 23 and 24 maintain the proposition locks, effectively ensuring that the same pigeons remain in the pigeonholes between time-steps.

The number of variables is small, dominated by the lock variables of which there are $4m(n+1)$ in a problem with 2^m pigeons and pigeonholes and $n+1$ states. The size of the formula in terms of clauses is dominated by the equivalences between the locks, which are $O(m(n+1))$.

5 Experimentation

Experiments were run on several domains to determine the effectiveness of the encoding. We hypothesised that:

- as the size of the problem increased, the PG-QBF approach would scale better than the SAT approach in both encoding time and solving time;
- as the size of the problem increased, PG-QBF would find solutions faster than the SAT approach;
- we would find problems that were too large to encode in SAT within the time limit allowed, but that could be encoded and solved using PG-QBF.

The domains selected for experimentation were the *pigeonhole*, *gripper* and *blocksworld* domains. These domains were chosen as they work well with ungrounded approaches. In other domains in which there is very little or no benefit from lifting the partially grounded QBF encoding resembles the SAT encoding.

	SAT		PG-QBF	
problem	encoding	solving	encoding	solving
pigeonhole2	0.04	0.00	0.04	0.00
pigeonhole4	0.09	0.00	0.04	0.00
pigeonhole8	0.33	0.12	0.06	0.00
pigeonhole16	5.53	5.06	0.1	0.09
pigeonhole32	155.71	*	0.13	0.58
pigeonhole64	-	-	0.16	22.8
gripper2	0.06	0.00	0.07	0.00
gripper4	0.13	0.03	0.1	0.01
gripper8	0.28	6.61	0.12	0.35
gripper16	1.22	1171.49	0.16	180.19
gripper32	7.03	-	0.3	-
blocksworld2	0.05	0.00	0.04	0.00
blocksworld4	0.11	0.01	0.09	0.02
blocksworld8	0.93	0.56	0.13	0.16
blocksworld16	13.43	20.47	0.18	1.16
blocksworld32	-	-	0.32	5834.88
blocksworld64	-	-	0.68	-

Table 1: Time taken to encode and solve problems using partially grounded QBF encodings and SAT based encodings. All times are in seconds. "-" means the time limit was reached, "*" means that the encoding ran out of memory.

For each domain a number of problems were generated, gradually increasing the size of the domain. These problems were then translated into SAT and PG-QBF encodings. The time taken for this translation was recorded. We used the SAT encoding used by SATPLAN'06 (Kautz, Selman, and

1. $\neg placed_0 \wedge empty_0 \wedge \neg in[0]_0 \wedge \neg in[1]_0$
2. $placed_{n+1}$
3. $place[1]_i \rightarrow place[0]_i$, for all $i = 1 \ldots (n+1)$
4. $place[2]_i \rightarrow place[0]_i$, for all $i = 1 \ldots (n+1)$
5. $place[1]_i \rightarrow (lock_j\_place[1]_i \leftrightarrow a_j)$, for all $i = 1 \ldots (n+1)$ and $j = 1 \ldots m$
6. $place[2]_i \rightarrow (lock_j\_place[2]_i \leftrightarrow a_j)$, for all $i = 1 \ldots (n+1)$ and $j = 1 \ldots m$
7. $place[0]_i \wedge \bigwedge_{j=1}^{m}(lock_j\_place[1]_i \leftrightarrow a_j) \rightarrow place[1]_i$, for all $i = 1 \ldots (n+1)$
8. $place[0]_i \wedge \bigwedge_{j=1}^{m}(lock_j\_place[2]_i \leftrightarrow a_j) \rightarrow place[2]_i$, for all $i = 1 \ldots (n+1)$
9. $place[1]_i \rightarrow \neg placed_i$, for all $i = 1 \ldots n$
10. $place[2]_i \rightarrow empty_i$, for all $i = 1 \ldots n$
11. $place[1]_i \rightarrow placed_{i+1}$, for all $i = 1 \ldots n$
12. $place[2]_i \rightarrow \neg empty_{i+1}$, for all $i = 1 \ldots n$
13. $place[1]_i \rightarrow (lock_j\_place[2]_i \leftrightarrow lock_j\_in[2]_{i+1})$, for all $i = 1 \ldots n$ and $j = 1 \ldots m$
14. $place[2]_i \rightarrow (lock_j\_place[1]_i \leftrightarrow lock_j\_in[1]_{i+1})$, for all $i = 1 \ldots n$ and $j = 1 \ldots m$
15. $empty_i \rightarrow place[2]_i \vee empty_{i+1}$, for all $i = 1 \ldots n$
16. $\neg empty_i \rightarrow \neg empty_{i+1}$, for all $i = 1 \ldots n$
17. $\neg placed_i \rightarrow place[1]_i \vee \neg placed_{i+1}$, for all $i = 1 \ldots n$
18. $placed_i \rightarrow placed_{i+1}$, for all $i = 1 \ldots n$
19. $in[1]_i \rightarrow in[1]_{i+1}$, for all $i = 1 \ldots n$
20. $\neg in[1]_i \rightarrow place[1] \vee \neg in[1]_{i+1}$, for all $i = 1 \ldots n$
21. $in[2]_i \rightarrow in[2]_{i+1}$, for all $i = 1 \ldots n$
22. $\neg in[2]_i \rightarrow place[2] \vee \neg in[2]_{i+1}$, for all $i = 1 \ldots n$
23. $in[1]_i \rightarrow (lock_j\_in[2]_i \leftrightarrow lock_j\_in[2]_{i+1})$, for all $i = 1 \ldots n$ and $j = 1 \ldots m$
24. $in[2]_i \rightarrow (lock_j\_in[1]_i \leftrightarrow lock_j\_in[1]_{i+1})$, for all $i = 1 \ldots n$ and $j = 1 \ldots m$

Figure 3: The constraints QBF instance Φ_n representing a pigeonhole problem with 2^m pigeons and pigeonholes, and $n+1$ states.

Hoffmann 2006) as it used the same STRIPS-based fluent/action representation as the PG-QBF encoding. Other SAT encodings, and additional constraints can also be used as a basis for partially grounded QBF encodings.

The sizes of these encodings can be seen in Table 2. This table highlights the difference in scaling between the approaches. The size of the SAT encoding very quickly becomes unreasonable, while the PG-QBF encodings scale much better; in the case of the pigeonhole problem PG-QBF encoding grows linearly with the number of objects.

The encodings were then solved using the SAT solver *picosat-535* and QBF solver *quantor-3.0*. Descriptions of these solvers can be found in Biere (2004) and (2008).

The times are recorded in Table 1. Times are presented in seconds, a hyphen indicating that the process was terminated after the time limit of 2 hours was reached. A star indicates that the process ran out of memory before the time limit was reached. The experiments were all run on a machine with 8GB of RAM and no artificial bound on the amount of memory used.

As can be observed, PG-QBF was able to encode and solve all of the pigeonhole instances considered, within the 2 hour limit, while SAT could not encode pigeonhole64, and was unable to solve pigeonhole32 and pigeonhole64 in the time available. PG-QBF was able to encode all of the instances across all domains in under one third of a second, while the time required by SAT to encode larger instances grew exponentially. Both approaches exhibit exponentially growing solution time, although the PG-QBF curve increases more slowly than the SAT curve. Neither approach was able to solve gripper32. These results support our first and second hypotheses, that PG-QBF scales better than SAT in both encoding and solution time, and that PG-QBF solves problems faster than SAT. Our third hypothesis is supported by pigeonhole64 and blocksworld32, which demonstrates that there are indeed instances that cannot be encoded by SAT, but can be encoded and solved by PG-QBF, within a fixed time limit.

6 Conclusions

In this paper we have introduced a method for lifting SAT encodings of planning problems into Quantified Boolean Formulae. Lifting objects into equivalence classes is a very powerful idea, being explored in different ways by a number of researchers in planning (Nguyen and Kambhampati 2001; Ridder and Fox 2011). It is well-known that grounding of

	SAT		PG-QBF	
problem	variables	clauses	variables	clauses
gripper2	120	657	129	488
gripper4	432	3267	290	1240
gripper8	1632	18183	643	2984
gripper16	6336	113679	1412	6952
gripper32	24960	782367	3077	15848
blocksworld2	72	261	109	331
blocksworld4	504	3891	352	1191
blocksworld8	3600	82503	963	3415
blocksworld16	26784	2265615	2422	8807
blocksworld32	-	-	5801	21415
blocksworld64	-	-	13468	50215
pigeonhole2	30	90	35	81
pigeonhole4	180	1232	79	207
pigeonhole8	1224	25008	177	501
pigeonhole16	8976	641696	399	1187
pigeonhole32	68640	18500160	901	2769
pigeonhole64	-	-	2027	6367

Table 2: Formula sizes for partially grounded QBF and SAT based encodings. "-" means the encoding ran out of time.

planning domains is infeasible for very large problems and that techniques for lifting planning instances can help with the solution of very large instances containing very large numbers of objects of the same type.

Our approach shows how to construct QBF encodings that both lift sets of similar objects into representative variables, and ensure consistent reasoning when the specific objects involved in transitions are not yet committed to. The approach is inspired by earlier work in least-commitment planning.

We have shown that our approach can lead to exponentially smaller encodings and allow larger problem instances to be solved than is possible using SAT.

References

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, 193–207.

Biere, A. 2004. Resolve and expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 59–70.

Biere, A. 2008. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*.

Cashmore, M., and Fox, M. 2010. Planning as QBF. *International Conference on Automated Planning and Scheduling Doctoral Consortium (ICAPS 2010)*.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2012. Planning as quantified boolean formulae. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*.

Dershowitz, N.; Hanna, Z.; and Katz, J. 2005. Bounded model checking with QBF. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, 408–414.

Ernst, M. D.; Millstein, T. D.; and Weld, D. S. 1997. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1169–1176.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4):189–208.

Huang, R.; Chen, Y.; and Zhang, W. 2012. SAS+ planning as satisfiability. *Journal of Artificial Intelligence Research* 43:293–328.

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

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, 359–363.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, 1194–1201.

Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*.

Mangassarian, H.; Veneris, A. G.; and Benedetti, M. 2010. Robust QBF encodings for sequential circuits with applications to verification, debug, and test. *IEEE Transactions on Computers* 59(7):981–994.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning.

Peschiera, C.; Pulina, L.; Tacchella, A.; Bubeck, U.; Kullmann, O.; and Lynce, I. 2010. The seventh QBF solvers evaluation (QBFEVAL'10). In *Proceedings of 13th the International Conference on Theory and Applications of Satisfiability Testing (SAT'10)*.

Ridder, B., and Fox, M. 2011. Performing a lifted reachability analysis as a first step towards lifted partial ordered planning. In *Proceedings of UK Planning and Scheduling SIG Workshop (Plansig'11)*.

Rintanen, J. 2001. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'01)*, 362–376.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2008. A compact and efficient SAT encoding for planning. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 296–303.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2009. SAT-based parallel planning using a split representation of actions. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4(2):177–192.

Stockmeyer, L. J., and Meyer, A. R. 1973. Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, 1–9.