University of Strathclyde

Planning as Quantified Boolean Formulae

MICHAEL CASHMORE

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

2013

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank all of the members of the Planning group at King's, for their support, and for teaching me in the first place. In particular I'd like to thank my supervisor, Maria Fox, for all of her input and guidance.

I would also like to thank those who collaborated with me, or lent me advice, especially Enrico Giunchiglia and Ian Gent, my external supervisor.

Finally I'd like to thank Bram Ridder and Daniele Magazzenni for sharing the best office and hundreds of cups of tea; Stephen Cashmore for fixing my grammar; and Michele Cashmore, my wife, for putting up with me while I wrote this thesis.

Abstract

This work explores the idea of classical Planning as Quantified Boolean Formulae. Planning as Satisfiability (SAT) is a popular approach to Planning and has been explored in detail producing many compact and efficient encodings, Planning-specific solver implementations and innovative new constraints. However, Planning as Quantified Boolean Formulae (QBF) has been relegated to conformant Planning approaches, with the exception of one encoding that has not yet been investigated in detail. QBF is a promising setting for Planning given that the problems have the same complexity.

This work introduces two approaches for translating bounded propositional reachability problems into QBF. Both exploit the expressivity of the binary-tree structure of the QBF problem to produce encodings that are as small as logarithmic in the size of the instance and thus exponentially smaller than the corresponding SAT encoding with the same bound. The first approach builds on the iterative squaring formulation of Rintanen; the intuition behind the idea is to recursively fold the plan around the midpoint, reducing the number of time-steps that need to be described from n to $log_2 n$. The second approach exploits domain-level lifting to achieve significant improvements in efficiency.

Experimentation was performed to compare our formulation of the first approach with the previous formulation, and to compare both approaches with comparative and state-of-the-art SAT approaches. Results presented in this work show that our formulation of the first approach is an improvement over the previous, and that both approaches produce encodings that are indeed much smaller than corresponding SAT encodings, in both terms of encoding size and memory used during solving. Evidence is also provided to show that the first approach is feasible, if not yet competitive with the state-of-the-art, and that the second approach produces superior encodings to the SAT encodings when the domain is suited to domain-level lifting.

Chapter 1

Introduction

1.1 Domain-Independent Planning

Domain-Independent Planning is a PSPACE-complete search problem [27]. A simple, abstract definition of Domain-Independent Planning splits the problem into two parts: a description of the initial and desired states of the world, in some formal language; and a set of actions–ways in which the world can be altered, perhaps by an agent. The first item forms the problem *instance*, while the second item is called the *domain*. A single domain may have many associated problem instances. The solution to a Planning problem is a set of actions that can be applied from the initial state in order to reach a desired state. This definition is very abstract and includes many varieties of formal languages and sub-problems, from non-deterministic and partially-observable worlds; to complex actions with conditional effects; to puzzles in a continuous real-time setting.

When regarding translations to Boolean formulae we will restrict ourselves to a simple subset of Planning problems. These Planning problems are specified using the standard STRIPS formulation [29]. A Planning problem is the 4-tuple $\langle \mathcal{F}, \mathcal{A}, I, G \rangle$ in which:

- $\mathcal{F}$ is a set of *fluents*, Boolean variables representing facts about the world;
- $\mathcal{A}$ is a set of *actions*;
- I is a formula over $\mathcal{F}$, describing the initial state of the world; and
- G is a formula over $\mathcal{F}$ representing the goal.

A fluent literal is the formula $\neg f$ or f where $f \in \mathcal{F}$. A valuation of $\mathcal{F}$ describes a *state*; this is denoted as S, a complete set of fluent literals over $\mathcal{F}$.

An action a is defined by $\langle P_a, E_a \rangle$ in which:

- P_a is a set of fluent literals, called the *preconditions*, and
- E_a is a set of fluent literals, called the *effects*.

An action a is applicable in a state S if $S \models P_a$. Applying a in a given state S leads to another state S^a such that $S^a \models E_a$ and $(f \leftrightarrow f^a) \quad \forall f \in S/E$, where $f^a \in S^a$ is the fact corresponding to f.

The solution to the problem is an ordered sequence of actions that are applicable from the initial state and end in a state that satisfies G. That is, $T : \{a_1, a_2, \ldots, a_n | (((I^{a_1})^{a_2}) \ldots)^{a_n} \models G\}$.

Both the lack of uncertainy in the initial state and the deterministic nature of actions are not true for Domain-Independent Planning in general; however, we make these assumptions for simplicity. The formulae presented in later sections deal only with deterministic action effects and unique initial states.

1.1.1 Domain definitions

Implementations of translations presented in later sections are from Planning Domain Definition Language, PDDL [67]. In this language the domain is described using *propositions*, *operators* and *object types*.

The problem instance defines the number of each *object type*. Binding specific objects to each parameter of a proposition (or operator) generates a fluent (or action). This process, called grounding, is described in detail below. Grounding all of the propositions generates the set of fluents $\mathcal{F}$ and similarly grounding the operators generates the set of actions $\mathcal{A}$.

For example, Figure 1.1 describes a domain for the problem of stacking blocks. This particular domain comes from the second International Planning Competition (IPC) [1]. A simple problem instance of this domain is shown in Figure 1.2, and graphically in Figure 1.3. This instance will be used as an example throughout this thesis.

1.1.2 Grounding

Traditionally, in order to translate the Planning problem into Boolean encodings we must first ground the instance. Grounding means generating fluents and actions from the propositions and operators of the domain. Briefly, the set of fluents are found by making every possible valid binding of objects to propositions. Actions are similarly created from the operators.

For example, the operator *stack* defined in Figure 1.1 using the problem instance described by Figure 1.2 would generate 9 actions. Each action would include a different pairing of blocks: $\mathit{stack}(A, A)$; $\mathit{stack}(A, B)$; $\mathit{stack}(A, C)$; and so on. Similarly the proposition *holding* will generate 3 different fluents: $\mathit{holding}(A)$; $\mathit{holding}(B)$; and $\mathit{holding}(C)$.

1.1.3 Parallel Optimised Planning

The efficiency of encoding Planning problems as Boolean formulae is greatly improved by the idea of parallel plans [8, 56]. In a parallel plan a number of actions can be applied simultaneously to a state. A *complex action* refers to a set of actions that are applied in parallel. The exact definition of a complex action depends upon the *parallel plan semantics* that are being used.

The length of the plan is its *makespan*, the number of timepoints at which complex actions are applied. The makespan is an important concept as it directly affects the size of the resulting formula after translation. A plan is said to be *makespan optimal* if the makespan is of size n, and no valid plan exists with a makespan of size $n - 1$.

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
               (ontable ?x - block)
               (clear ?x - block)
               (handempty)
               (holding ?x - block)
               )

  (:action pick-up
           :parameters (?x - block)
           :precondition (and (clear ?x)
                              (ontable ?x)
                              (handempty))
           :effect
           (and (not (ontable ?x))
                (not (clear ?x))
                (not (handempty))
                (holding ?x)))

  (:action put-down
           :parameters (?x - block)
           :precondition (holding ?x)
           :effect
           (and (not (holding ?x))
                (clear ?x)
                (handempty)
                (ontable ?x)))
  (:action stack
           :parameters (?x - block ?y - block)
           :precondition (and (holding ?x) (clear ?y))
           :effect
           (and (not (holding ?x))
                (not (clear ?y))
                (clear ?x)
                (handempty)
                (on ?x ?y)))
  (:action unstack
           :parameters (?x - block ?y - block)
           :precondition (and (on ?x ?y)
                              (clear ?x)
                              (handempty))
           :effect
           (and (holding ?x)
                (clear ?y)
                (not (clear ?x))
                (not (handempty))
                (not (on ?x ?y)))))
```

Figure 1.1: The domain for blocksworld, as used in IPC2.

```
(define (problem BLOCKS-EXAMPLE)
(:domain BLOCKS)
(:objects A B C - block)
(:init (clear A) (clear B) (ontable B) (ontable C)
       (on A C) (handempty))
(:goal (AND (on A B) (on B C))))
```

Figure 1.2: A simple instance of blocksworld.

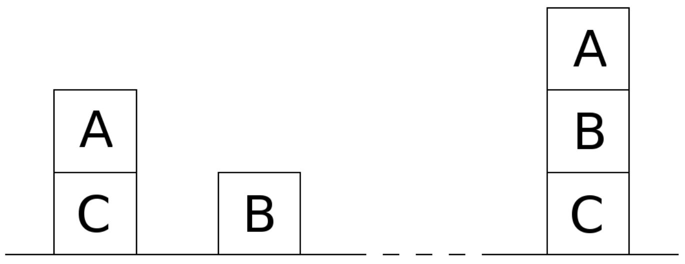


Figure 1.3: The initial state (left) and goal state (right) of the example blocksworld problem.

There are a number of different definitions for parallel plans. Important to Planning as Boolean formulae are $\forall$-*step semantics*, *process semantics* and $\exists$-*step semantics*.

In $\forall$-step semantics actions can be applied in parallel as long as they could be applied sequentially in any total order terminating in the same state. So, for every total ordering $a_1, a_2, \ldots, a_m$ of complex action A:

$$(((S_i^{a_1})^{a_2})\ldots)^{a_m} \equiv S_{i+1}.$$

For STRIPS this can be achieved by applying in parallel only actions that are not *mutually exclusive*. Two actions, $\langle P_1, E_1\rangle, \langle P_2, E_2\rangle \in \mathcal{A}$ are mutually exclusive if $\exists f \in \mathcal{F}$ such that:

- $\neg f \in E_1$ and $f \in E_2$;
- $\neg f \in E_2$ and $f \in E_1$;
- $\neg f \in E_1$ and $f \in P_2$;
- $\neg f \in E_2$ and $f \in P_1$;
- $f \in E_1$ and $\neg f \in P_2$; or
- $f \in E_2$ and $\neg f \in P_1$.

A $\forall$-step plan is a sequence $T : \{A_1, A_2, \ldots, A_n\}$ of complex actions, in which each set forms an independent set with respect to mutual exclusivity.

A process plan is a $\forall$-step plan $T : \{A_1, A_2, \ldots, A_n\}$ in which there is no $i \in 2, 3, \ldots, n$ and $a \in A_i$ such that $T : \{A_1, A_2, \ldots, A_{i-1} \bigcup\{a\}, A_i \backslash \{a\}, \ldots, A_n\}$ is also a valid plan. The earliest appearance semantics reduces the number of valid plans, but does not affect plan existence. This can easily be seen in that for every $\forall$-step plan with makespan n a plan valid under process semantics can be obtained by repeatedly moving actions violating the condition one time point earlier. The idea of process semantics has been explored in other problems [3, 23, 45] and formalised in Planning [90].

In $\exists$-step semantics actions can be applied in parallel as long as there exists at least one way in which they could be applied sequentially under a total order. This definition allows for much greater parallelism, and possibly shorter makespans. An important difference from $\forall$-semantics is that the result of applying a complex action A to a state S_i does not lead to a unique state S_{i+1}. It is necessary to make the order of operators implicit in their descriptions. The idea was proposed by Dimopoulos et al. [24] and later formalised [90].

The formulae presented here use $\forall$-step semantics, although the choice of parallel plan semantics is orthogonal to the choice between different SAT and QBF translations.

1.1.4 Planning as Satisfiability

Planning as SAT is one of the best known and effective techniques for classical Planning: SATPLAN [55] 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 (SAT) formula obtained by unfolding, n times, the symbolic transition relation of the automaton described by the Planning problem.

In the following, we use X to denote the whole set of variables, i.e., $\mathcal{F} \bigcup \mathcal{A}$. The SAT encoding of a Planning problem is a 4-tuple $\langle I, \sigma, \tau, G \rangle$ where

- I is a an interpretation of $\mathcal{F}$ and represents the *initial state*;
- σ is a Boolean formula over X that represents the *state constraints* within the automaton;
- τ is a Boolean formula over $X \cup X'$ where $X' = \{x' : x \in X\}$ is a copy of the set of variables and represents the *transition relation* of the automaton describing how (complex) actions affect states (we assume $X \cap X' = \emptyset$);
- G is a Boolean formula over $\mathcal{F}$ and represents the set of *goal states.*

The only assumption that we make is that the description is deterministic: there is only one state satisfying I and the execution of a (complex) action α in a state S can lead to at most one state S'. More formally, for each state S and complex action α there is at most one interpretation extending $S \cup \alpha$ and satisfying τ.

Consider a Planning problem $\langle I, \sigma, \tau, G \rangle$. As standard in Planning as SAT, the existence of a plan with makespan n (or lower) is proved by building a propositional formula with n copies of the set of variables. 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 $x_\beta \in X_\beta$.

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

$$I(X_1) \wedge \bigwedge_{i=1}^{n+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge G(X_{n+1}) \qquad (n \geq 0) \tag{1.1}$$

and a *plan* for Π_n is an interpretation satisfying (1.1).

1.1.5 Example of Planning as Satisfiability

Consider the simple Planning problem in Figure 1.1 and Figure 1.2. Described in this subsection is one example of encoding a Planning problem as a Boolean formula. After grounding we obtain the set $X : \mathcal{F} \bigcup \mathcal{A}$, which contains variables representing the 19 fluents:

$$\begin{gathered}(\textit{on}\,A\,A),\ (\textit{on}\,A\,B),\ (\textit{on}\,A\,C),\ (\textit{on}\,B\,A),\ (\textit{on}\,B\,B),\\ (\textit{on}\,B\,C),\ (\textit{on}\,C\,A),\ (\textit{on}\,C\,B),\ (\textit{on}\,C\,C),\\ (\textit{ontable}\,A),\ (\textit{ontable}\,B),\ (\textit{ontable}\,C),\\ (\textit{clear}\,A),\ (\textit{clear}\,B),\ (\textit{clear}\,C),\\ (\textit{holding}\,A),\ (\textit{holding}\,B),\ (\textit{holding}\,C),\\ (\textit{handempty})\end{gathered}$$

and 24 actions:

$$\begin{gathered}(\textit{stack}\,A\,A),\ (\textit{stack}\,A\,B),\ (\textit{stack}\,A\,C),\ (\textit{stack}\,B\,A),\ (\textit{stack}\,B\,B),\\ (\textit{stack}\,B\,C),\ (\textit{stack}\,C\,A),\ (\textit{stack}\,C\,B),\ (\textit{stack}\,C\,C),\\ (\textit{unstack}\,A\,A),\ (\textit{unstack}\,A\,B),\ (\textit{unstack}\,A\,C),\ (\textit{unstack}\,B\,A),\ (\textit{unstack}\,B\,B),\\ (\textit{unstack}\,B\,C),\ (\textit{unstack}\,C\,A),\ (\textit{unstack}\,C\,B),\ (\textit{unstack}\,C\,C),\\ (\textit{put-down}\,A),\ (\textit{put-down}\,B),\ (\textit{put-down}\,C),\\ (\textit{pick-up}\,A),\ (\textit{pick-up}\,B),\ (\textit{pick-up}\,C).\end{gathered}$$

The encoding of this problem is the 4-tuple $\langle I, \sigma, \tau, G \rangle$.

1. For set of fluents $F_0 \subseteq X$,

   $$F_0 : \{ \begin{array}{c}(\textit{on}\,A\,C),\ (\textit{ontable}\,B),\ (\textit{ontable}\,C),\\ (\textit{clear}\,A),\ (\textit{clear}\,B),\ (\textit{handempty})\end{array} \}$$

   $I(X) \models F_0$ and $I(X) \models \neg f$, $\forall f \notin F_0$.

2. $\sigma(X)$ models the mutual exclusion relations and the action preconditions:

   - $\sigma(X) \models \neg a_1 \vee \neg a_2$, $\forall a_1, a_2 \in X$ such that a_1 and a_2 are mutually exclusive; and
   - $\sigma(X) \models a \rightarrow p$, $\forall p \in P_a$, for example:

     $$(\textit{unstack}\,A\,C) \rightarrow (\textit{on}\,A\,C) \wedge (\textit{clear}\,A) \wedge (\textit{handempty}).$$

3. $\tau(X_\alpha, X_\beta)$ ensures that the effects of an action applied in X_α are present in X_β, and that a fluent made true in X_β implies a supporting action, or fact in X_α. More formally:

   - $\tau(X_\alpha, X_\beta) \models a \rightarrow e, \forall e \in E_a$ for each $a \in X_\alpha$ and $e \in X_\beta$; and
   - $\tau(X_\alpha, X_\beta) \models f_\beta \rightarrow (f_\alpha \vee A_f)$, for each $f \in \mathcal{F}$, where f_α represents the copy of f in set X_α and similarly for f_β. $A_f \subseteq X_\alpha$ represents the achievers of f, that is, all actions a such that $f \in E_a$.

   For example, consider action $(\textit{pick-up}\,A)$ and fact $(\textit{holding}\,A)$. $\tau(X_\alpha, X_\beta)$ would contain the constraints:

   $$(\textit{pick-up}\,A)_\alpha \rightarrow \\ (\textit{holding}\,A)_\beta \wedge \neg(\textit{ontable}\,A)_\beta \wedge \neg(\textit{clear}\,A)_\beta \wedge \neg(\textit{handempty})_\beta$$

   and

   $$(\textit{holding}\,A)_\beta \rightarrow (\textit{holding}\,A)_\alpha \vee (\textit{pick-up}\,A)_\alpha \\ \vee(\textit{unstack}\,A\,A)_\alpha \vee (\textit{unstack}\,A\,B)_\alpha \vee (\textit{unstack}\,A\,C)_\alpha.$$

4. $G(X) \models F_g$ for the goal facts $F_g\{(\textit{on}\,A\,B), (\textit{on}\,B\,C)\}$.

It is easy to check that there exists a plan for the problem for any makespan $n \geq 6$. Indeed, from our definitions, it is enough to construct the propositional formula (1.1) for a sufficiently large n and then check its satisfiability.

The encoding used here is an example of a simple state-based encoding with $\forall$-step semantics first presented by Kautz and Selman [55]. Other encodings will be reviewed in Chapter 2.

1.2 Quantified Boolean Formulae

Quantified Boolean Formulae (QBF), which is PSPACE-complete [93, 94], 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), which is a conjunction of disjunctions of literals. These literals are instances of Boolean variables in either positive or negative phase. The expression is prefixed by a quantifier layer in which every variable present in the expression is quantified either existentially or universally.

The decision problem is stated as: given φ, a Boolean expression in CNF, 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? In other words,

$$\exists X_1, \forall X_2, \exists X_3, \ldots, \exists X_m\, \varphi?$$

For example:

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

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 leaves. The outermost quantifier can be expanded, removing it from the problem. Where $\phi[x/\top]$ is the result of assigning $x \rightarrow \mathit{true}$ 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.

This idea of expansion is useful in various proofs, especially in Chapter 3. 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:

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

Expanding every universally quantified variable in the problem will produce its existential closure, flattening the formula to an exponentially larger SAT instance.

1.3 Quantified Boolean Formulae in Planning

An encoding of reachability into QBF that is logarithmic in the number of timesteps was described by Rintanen [80] as a motivating example. Consider the formula $\mathit{reach}_n(I, G)$ that represents a transition sequence of length 2^n between two states I and G. This formula could encode problems as QBF by recursively folding the transition sequence around the midpoint:

$$\begin{array}{rl} \mathit{reach}_o(I, G) & := \tau(I, G) \\ \mathit{reach}_{n+1}(I, G) & := \exists S \forall y(\\ & \qquad \neg y \rightarrow \mathit{reach}_n(I, S) \wedge y \rightarrow \mathit{reach}_n(S, G) \quad). \end{array}$$

This encoding is formalised in Chapter 3 as the Flat Encoding.

QBF has also been considered in the context of conformant Planning [79], but not as a practical approach to classical Planning until now. Conformant Planning is the problem of finding conditional plans in a setting where the initial state is unknown, only partially known, or action effects are non-deterministic [43]. We can assume that all action effects are deterministic and all uncertainty is contained in the initial state without any loss of expressivity.

With such an assumption, a Conformant Planning problem is the 4-tuple $\langle \mathcal{F}, \mathcal{A}, I, G \rangle$ in which:

- $\mathcal{F}$ is a set of observable *fluents* that decide how plan execution proceeds;
- $\mathcal{A}$ is a set of *actions*;
- I is a formula over $\mathcal{F}$, describing the set of initial states; and
- G is a formula over $\mathcal{F}$ representing the goal.

A conditional plan determines for all possible initial states an execution that reaches a state $S \models G$.

Encodings of Conformant Planning as QBF use the greater expressive power to quantify universally over the initial state. In this way conditional plans are found with executions that find a state satisfying the goal for any possible initial state. Encodings of this type are discussed further in Chapter 2.

QBF has not been seriously considered for use in classical Planning because it has not been evident that its expressive power can be exploited when all parts of the problem are determined. Also, SAT has produced excellent results for classical Planning and there has not been a strong drive to find ways to improve on these using richer SAT methods. Given the maturity of research into SAT solving compared to solving QBF it is not surprising that straighforward translations of Planning problems into QBF yield very poor performance. However, SAT-based Planning, though quite successful, suffers from the weakness that it is easy to come up with problems in which the number of steps required, or number of grounded variables, is large, making it impossible to even encode the original problem as a propositional formula. The same problem arises in bounded model checking [7].

We propose that QBF, being PSPACE-complete, is a more natural candidate for translation. This has been proposed before as a possibility for overcoming these shortcomings of the SAT-based approach [22, 52, 63]. In particular, Jussila and Biere [52] and Rintanen [80] present an encoding that is logarithmic in the makespan, resembling the proof of the PSPACE-hardness of solving QBFs [93, 94]. However, no practical encoding method has yet been proposed. While obvious translations do not work, the idea explored in this thesis is that QBF encodings can lead to more efficient solution than SAT encodings when the right encoding is used.

1.4 Statement of Thesis

This thesis explores the claim that QBF encodings lead to more efficient solution of Planning problems than SAT encodings *when the right encoding is used.* We present two such translations with the potential to be exponentially smaller than the equivalent SAT translation of the same makespan. The first translation is based upon the iterative squaring formulation of Rintanen [80], folding the formula recursively from the mid-time-step, while the second exploits the universal quantification to describe a plan in a manner that is only partially grounded.

We describe experiments performed on these encodings to determine their effectiveness, and present results showing that the first translation is a marked improvement upon the previous formulations and that, while still not yet competitive with SAT-based techniques, the translation has the potential to be a far more scalable approach to Planning as Boolean formulae. The competitiveness result is not surprising, given the relative maturity of SAT solving compared to QBF. We also present results that show our second approach is competitive with SAT-based techniques on domains that benefit from domain level lifting, finding problems that are practically impossible to encode in SAT and consequently are unsolvable, but which can be encoded and solved using our QBF approach.

Chapter 2

Background

2.1 Planning as Satisfiability

Planning as Satisfiability was pioneered by Kautz and Selman, beginning with a translation from Planning into propositional satisfiablity [55]. This section will outline further evolutions and approaches in this field. The contributions described here mainly fall into one of three categories: an alternate encoding of state or transition relation; the embedding of additional, or alternate, Planning-specific knowledge; or Planning-specific improvements to the SAT solver. Many of these ideas are orthogonal to the choice of representation between SAT and QBF; improvements to Planning as SAT are applicable, usually directly, to Planning as QBF. After a short description of the beginnings of Planning as Satisfiability, contributions to the field are described, and their relevance to this work outlined, in three subsections corresponding to these categories.

Kautz and Selman first presented a translation from Planning to propositional satisfiability as a complementary approach to deductive systems [55]; stating as their idea:

> *In the Planning as satisfiability approach, a Planning problem is not a theorem to be proved; rather, it is simply a set of axioms with the property that any model of the axioms corresponds to a valid plan.*

The encoding they presented had a one-to-one correspondence between the fluents and actions at each time-step and the variables in the SAT formula. The constraints of the problem were:

- an action implies both its preconditions and effects;
- exactly one action occurs at each time-step;
- the initial state is completely specified; and
- classical frame axioms hold for all actions.

Classical frame axioms state that if an action does not change the truth value of a fluent then the fluent remains true or remains false when the action occurs. The number of classical frame axioms becomes very large as the number of operators is increased [54] and later encodings, including those presented in this paper, move away from them, for that reason.

Kautz and Selman 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 $(move\, x\, y\, z)_i$ operator for moving a block x directly from the top of block y to the top of block z at time-step i would be replaced by three split operators: $(move\, x)_i^{obj}$, $(move\, y)_i^{src}$ and $(move\, z)_i^{dest}$. Kautz and Selman liken this procedure to "lifting" the Planning problem, and the same idea forms the basis of the encoding presented in Chapter 4: a Partially Grounded QBF Encoding, and in the encoding by Robinson et al. [92] presented later in this section.

2.1.1 Alternate Encodings of Planning Problems

Encodings here are presented in roughly chronological order with reference to the ways in which they relate to the QBF encodings in Chapters 3 and 4. The current state-of-the-art in terms of Planning as SAT are the relaxed $\exists$-step encodings of Wehrle and Rintanen [96], the Split Representation of Robinson et al. [92], and the SAS^+-based encoding of Huang et al. [50, 51]. All three approaches are directly applicable to encodings described in this work, and the third bears very strong parallels to the encoding in Chapter 4.

Graphplan-based Encodings

The Graphplan system from Blum and Furst [8] instigated a new approach to encoding Planning problems as SAT. The Graphplan system converts a STRIPS-style Planning problem into a Planning graph; a directed, levelled graph with alternating layers of nodes. Edges in the graph only exist between nodes in adjacent layers. The layers alternate between fluent and action layers, the nodes of which represent fluents and actions respectively. Each layer is indexed by the time-step for that layer. For example, the first layer contains fluents true at time 1, the second contains possible actions at time 1, and the third layer contains fluents that are possible at time 2. For every fluent there also exists a no-op "maintain" operator that simply has that fluent as a precondition and effect.

A node is added to the first layer for each fluent that is true in the initial state; all possible actions whose preconditions exist in the previous layer are added; and all add effects of actions in layer i are added to layer $i+1$. Edges in the explicitly represent relations between actions and propositions. Each action node in level i is connected by "precondition edges" to their preconditions in layer $i-1$, by "add-edges" to their add effects, and by "delete-edges" to their delete effects in layer $i+1$. The plan graph reaches a fix point when the new layer is identical to the previous.

The constraints upon the Planning graph are weaker than those imposed upon a valid plan. In particular, an action layer contains all possible actions applicable at that time. The advantage is that the Planning graph can be constructed relatively quickly, in polynomial time. Planning graphs have the important property that:

> If a valid plan exists using n or fewer time-steps, a valid plan exists as a subgraph of a Planning graph with n action layers.

A solution is a subgraph that contains in the final layer a set of fluents that satisfies the goal, contains no two mutually exclusive actions in the same layer, for each fluent node contains at least one supporting action node in the previous layer (unless it is in the first layer), and for each action node contains every fluent node comprising its preconditions in the previous layer. This solution belongs to the family of $\forall$-step semantic plans described in Section 1.1.3.

An integral part of the Planning graph is to compute mutual exclusion relations between fluents and actions. Not all relations are discovered, specifically only those which are noticed by the following rules, which extend those described in section 1.1.3:

- Interference: Two actions a and b are mutually exclusive if there exists fluent f such that $\neg f \in E_a$ and $(f \in P_b) \vee (f \in E_b)$. This corresponds to the notion of mutual exclusivity already introduced.
- Competing Needs: Two actions a and b are also mutually exclusive if there exist two fluents f_1 and f_2 such that $f_1 \in P_a$ and $f_2 \in P_b$ and f_1 and f_2 are mutually exclusive in the previous layer.
- Fluent mutual exclusions: Two fluents f_1 and f_2 are mutually exclusive if all actions which achieve f_1 are mutually exclusive with all actions that achieve f_2 in the previous layer.

Graphplan is faster on many domains than the linear encodings of SATPLAN [58]. Kautz and Selman speculated that this was partly due to the mutual exclusion computations performed by Graphplan, pruning nodes from the graph during instantiation. Noting that a Planning graph is very similar to their linear encodings Kautz et al. [54, 58] were able to automatically convert Planning graphs into CNF notation. Kambhampati [53] describes the encoding as posing the extraction of a subgraph as a SAT problem.

The advantage of this formulation is that the classical frame axioms are no longer required. Instead each fluent implies the disjunction of possible causes. This is an example of *explanatory* frame axioms, which say that when the truth value of a fluent changes, then one of the actions which adds or deletes it must have occurred. While classical frame axioms require $O(n|\mathcal{A}||\mathcal{F}|)$ clauses, explanatory frame axioms require only $O(n|\mathcal{F}|)$ clauses. Although the clauses are larger, they are limited by the number of causes that explain the change in a fluent's truth value and in practice the size of the encodings is much reduced [56]. Consequently, later encodings, including the encodings presented here, use explanatory frame axioms.

The resulting system, called BLACKBOX [57], was able to automatically create encodings from PDDL by:

1. generating a Planning graph to the layer at some fixed level k;
2. converting the Planning graph into CNF, and running a general simplifying algorithm on the result;
3. solving the SAT problem with a fast SAT solving system;
4. converting a model, if one is found, into the corresponding plan; otherwise, incrementing k and repeating the process.

Blackbox came first in the 1st IPC in 1998 [68].

The encodings in this work use a Planning graph as an initial step, implemented in an efficient fashion [34]. The plan graph is used for its mutual exclusion computations and also to generate a set of reachable fluents and actions, pruning large numbers of variables before translation.

State- and Action-based Encodings

Kautz et al. introduced the idea of state- and action-based encodings [54, 56] as a refinement of their linear and Graphplan-based encodings that enjoys the advantages of both approaches. The key idea behind the encodings is that they emphasise the use of axioms to assert the constraints upon individual states, giving a secondary role to the axioms describing operators. This step is inherent in every encoding presented in this work; the example SAT encoding (formula (1.1)) presented in Section 1.1.4 makes a clear separation between the state constraints (σ) and the transition relation (τ). Behind this decision is the reasoning that a well-defined, internally consistent state requires only a small number of axioms to define the transition relation.

These encodings, while no more compact than Graphplan encodings, can be significantly simplified. The arity of operators can be reduced, as described above. Also, variables can be compiled away, to the extreme of compiling away all of the variables representing either the actions or the fluents. The second of these, a purely action-based encoding, was used as the basis for SATPLAN'04, a successor to Blackbox. SATPLAN'04 came first in the 4th IPC in 2004 [48].

Consider the fluents $(on\,A\,B)$, $(clear\,A)$, and $(holding\,A)$ along with the action $(unstack\,A\,B)$. In an encoding using explanatory frame axioms and both fluent and action variables, the constraints upon the preconditions and support for these fluents are separate:

$$\begin{gathered}(holding\,A)_{i+1} \rightarrow (unstack\,A\,B)_i \vee (holding\,A)_i \\ \wedge \\ (unstack\,A\,B)_i \rightarrow (on\,A\,B)_i \wedge (clear\,A)_i.\end{gathered}$$

Compiling away the action variables, in this case $unstack(A, B)$, we instead use a single constraint that relates the fluent $holding(A)$ with $clear(A)$ and $on(A, B)$:

$$(holding\,A)_{i+1} \rightarrow ((on\,A\,B)_i \wedge (clear\,A)_i) \vee (holding\,A)_i.$$

This technique, while reducing the number of variables significantly, obviously increases the size of the constraints. Consider the simple Blocksworld example and imagine that there are several more blocks for A to be resting upon – already the constraint involves a large disjunction of conjunctions. Converting this into conjunctive normal form would create a large encoding indeed. Kautz et al. make the observation that compiling away actions can lead to an exponential blow-up in the size of the encoding, but compiling away fluents gives only a polynomial (in $|Domain|$) increase in size [54]. This should be obvious as the Graphplan no-op actions implicitly encode the fluent information.

Despite the increase in size of state-based encodings, experimental results were encouraging [56], and action-based encodings proved to be even better in SATPLAN'04. The encodings presented in Chapter 3 both use an action-only encoding.

SATPLAN'06 [59] – the updated version of SATPLAN'04 – uses both fluent and action variables, encodes only a subset of the mutual exclusion constraints and performs some post-processing on the solution to remove some of the unnecessary actions. SATPLAN'06 came first in the 5th IPC in 2006 [36].

Lifted Causal Encodings

Lifted causal encodings were first introduced by Kautz et al. [54] alongside their Graphplan-based encodings, inspired by the lifted version of the SNLP causal link planner of McAllester and Rosenblitt [66].

The lifted causal encoding is completely different from the state-based encodings in that there is no proper notion of a state. Kautz et al. [54] introduced two encodings, the first a translation of ground causal Planning to SAT, which is not nearly as concise as the lifted encoding, and the second a translation of lifted causal Planning to lifted SAT, which is then reduced to SAT.

Ground causal Planning involves all grounded actions of the domain, two actions representing the initial and goal states and a set of plan steps $o_0 \ldots o_n$. The initial state is represented by an action a with $E_a \equiv I$, the goal state is represented by an action a with $P_a \equiv G$. A ground causal link is an assertion of the form $o_i \rightarrow^p o_j$ where o_i and o_j are plan steps. The assertion is true if $p \in P_{o_j}$; $p \in E_{o_i}$; and there is no step between o_i and o_j that either adds or deletes p.

A causal plan is an assignment of ground actions to plan steps such that:

1. If action a is assigned to a plan step o_i there exists a causal link of the form $o_j \rightarrow^p o_i$ for all $p \in P_a$.
2. Every causal link is true: $o_j \rightarrow^p o_i$ implies that $o_j < o_i$ and any plan step o_k, $k \neq i, j$, assigned an action that adds or deletes p, $(o_k < o_j) \vee (o_i < o_k)$.
3. The plan steps can be placed in a total order that does not violate any ordering constraints.

This can be translated into SAT using the constraints described in Figure 2.1. The size of the encoding (in variables) is dominated by the causal link variables, of which there are $O(n^2|\mathcal{F}|)$ and the *assign* variables of which there are $O(n|A|)$. The size of the resultant formula is dominated by size of constraint 7 which is $O(n^3|\mathcal{F}|)$.

The lifted SAT problem consists of first-order clauses – disjunctions of first-order literals, possibly containing free variables. The problem is satisfiable if there is a ground substitution such that the resulting SAT problem is satisfiable. For example:

$$\begin{array}{l} P(x) \rightarrow Q(x) \\ Q(x) \rightarrow W(x) \\ P(y) \\ x = a \vee x = b \\ P(a), \neg W(a), P(b), \neg W(b) \end{array}$$

is unsatisfiable, because x must be a or b, and in either case one of the first two clauses must be violated.

Kautz et al. described a translation from Planning to lifted SAT without grounding, followed by a reduction from lifted SAT to SAT. The result of these two translations produced their most compact encoding.

1. $assign(a_1, o_i) \vee \ldots \vee assign(a_m, o_i)$ for all $i = 1 \ldots n$
2. $\neg(assign(a, o_i) \wedge assign(b, o_i))$ for all $i = 1 \ldots n$ and $a, b \in A | a \neq b$.
3. $\neg adds(I, p)$ for all $p \in \mathcal{F} - I$
4. $needs(G, p)$ for all $p \in G$
5. $needs(o_i, p) \rightarrow (o_1 \rightarrow^p o_i \vee \ldots \vee o_n \rightarrow^p o_i)$ for all $i = 1 \ldots n$ and $p \in \mathcal{F}$
6. $o_i \rightarrow^p o_j \rightarrow adds(o_i, p)$ for all $i = 1 \ldots n-1$; $j = 2 \ldots n$; and $p \in \mathcal{F}$
7. $o_i \rightarrow^p o_j \wedge dels(o_k, p) \rightarrow (o_k < o_i \vee o_j < o_k)$ for all $i = 1 \ldots n-1$; $j = 2 \ldots n$; $k \neq i, j$; and $p \in \mathcal{F}$
8. $\neg(o_i < o_i)$ for all $i = 1 \ldots n$
9. $o_i < o_j \wedge o_j < o_k \rightarrow o_i < o_k$ for all $i, j, k = 1 \ldots n$

Figure 2.1: Constraints for the translation of grounded causal Planning to SAT from Kautz et al. [54].

The variables included in the encoding consist of:

- n sets denoted $A_1 \ldots A_n$, copies of every operator in the domain for each Planning step o_i as ungrounded first-order functions, the variables of which are disjoint from all operations appearing in other plan steps;
- $assign(op, o)$ for each operator $op \in A_i$ and plan step $o \in O$;
- $Pre(A_i)$, $Add(A_i)$ and $Del(A_i)$ for each plan step, representing all the preconditions, add effects and delete effects of the set of actions. $Pre(A_i)$ denotes all of action A_i's preconditions while $Pre(o_i, p)$ denotes that fluent p is a precondition of the action assigned to step o_i.

$Vars(A)$ denotes the set of all variables appearing in $A_I \bigcup A_1 \bigcup \ldots \bigcup A_n \bigcup A_G$ and Dom is the set of all objects in the domain with which the variables are substituted.

The constraints involved are described by Figure 2.2.

Satisfying constraints 1 and 2 are analogous to choosing which operator to apply in each Planning step, then objects are bound to the parameters of the operator by satisfying constraint 3. Constraints 4 and 5 assert the preconditions of each action. Constraint 6 selects the causal source of every precondition. Constraints 7 and 8 ensure that the causal source is ordered before the precondition and contains the add effect, while constraints 9, 10 and 11 select equations between elements in the add and precondition lists. For example we might have:

$$Adds(o_1, (clear\, x_3)) \rightarrow ((clear\, x_3) = (clear\, y_1))$$

with the actions:

$$(unstack\, x_1\, y_1)$$
$$(pick\text{-}up\, x_3).$$

1. $assign(a_1, o) \vee \ldots \vee assign(a_n, o)$ for all $o \in O$
2. $\neg(assign(a, o) \vee assign(b, o))$, for all $o \in O$ and $a, b \in A_o$ with $a \neq b$
3. $x = c_1 \vee \ldots \vee x = c_m$, for all $x \in Vars(A)$ and $c_i \in Dom$
4. $assign(a, o) \rightarrow Pre(o, p)$ for all $o \in O$; $a \in A_o$; and $p \in Pre(A_o)$
5. $Pre(G, p)$ for all $p \in G$
6. $Pre(o_i, p) \rightarrow (I \rightarrow^p o_i \vee o_1 \rightarrow^p o_i \vee \ldots \vee o_n \rightarrow^p o_i)$ for all $o_i \in O \bigcup G$ and $p \in Pre(A_i)$
7. $o_i \rightarrow^p o_j \rightarrow o_i < o_j$ for all $o_i \in O \bigcup I$; $o_j \in O \bigcup G$; and $p \in Pre(A_j)$
8. $o_i \rightarrow^p o_j \rightarrow Adds(o, p)$ for all $o_i \in O \bigcup I$; $o_j \in O \bigcup G$; and $p \in Pre(A_j)$
9. $assign(o, a) \wedge Adds(o, p) \rightarrow (p = q_1 \vee \ldots \vee p = q_m)$ for all $o \in O$; $a \in A_o$; $p \in Pre(A_o \bigcup G)$; and $q_i \in Adds(a)$
10. $Adds(I, p) \rightarrow (p = q_i \vee \ldots \vee p = q_m)$ for all $p \in Pre(A)$ and $q_i \in I$
11. $\neg Adds(I, p)$ for all $p \in \mathcal{F} - I$
12. $assign(a, o) \wedge (p = q) \rightarrow Dels(o, p)$ for all $o \in O$; $a \in A_o$; $p \in Pre(A_o \bigcup G)$; and $q \in Dels(a)$
13. $o_i \rightarrow^p o_j \wedge Dels(o_k, p) \rightarrow (o_k < o_i \vee o_j < o_k)$ for all $o_i \in O \bigcup I$; $o_j \in O \bigcup G$; $o_k \in O - o_i, o_j$; and $p \in Pre(A_o)$
14. $o_i < o_j \wedge o_j < o_k \rightarrow o_i < o_k$ for all $o_i, o_j, o_k \in O \bigcup I, G$
15. $\neg(o_i < o_i)$ for all $o_i \in O \bigcup I, G$

Figure 2.2: Constraints for the translation of lifted causal Planning to lifted SAT from Kautz et al. [54].

Once the equations are selected the truth of these equalities is checked by the semantics of lifted SAT. In the example the equation $((clear\, x_3) = (clear\, y_1))$ ensures that the identity of the blocks substituted into y_1 and x_3 is the same.

Constraints 12 and 13 assert the delete effects and ensure they do not interfere with causal links. Finally constraints 14 and 15 ensure that there exists a valid totally ordered plan from the partially ordered plan steps.

The size of every constraint set is either linear or quadratic in the number of plan steps, with the exception of constraints 13 and 14, which are cubic.

Kautz et al. follow up the translation with a reduction into SAT that yields a polynomial sized encoding [54].

The lifted causal encoding bears important similarities to the encoding presented in Chapter 4. Both encodings encode operators and predicates without grounding, using the expressive power of their respective target formalisms to bind the parameters to objects implicitly. The Partially Grounded QBF encoding uses QBF rather than lifted SAT to provide the necessary expressivity and encodes the plan in a total order, resulting in a combination between the lifted causal encoding and state/Graphplan-based encodings.

∃-step Semantics

∃-step semantics have been used in recent encodings by Wehrle and Rintanen [84, 85, 86, 96]. The idea is based on the work of Dimopoulos et al. in encoding Planning problems as nonmonotonic logic programs [24]. Dimopoulos et al. drew on the ideas from GRAPHPLAN [8] and SATPLAN [58] to generate a translation from Planning to nonmonotonic logic with the *stable model* semantics [35] of SMODELS [73].

In their encoding, Dimopoulos et al. introduced the idea of a parallel plan being *post-serializable*. A set of actions in a parallel plan is post-serializable if there exists at least one order in which they can be applied without violating any preconditions. By exploiting this structure it is possible in some domains to gain much more parallelism than the ∀-step semantics of previous encodings. The post-serialization property can be described more formally using the *preconditions-effects graph* of a set of actions A.

The graph, A_G, is a directed graph that contains a node for each action in A, and an edge $\langle a_i, a_j \rangle$ exists if the preconditions of a_i are inconsistent with the effects of a_j. The set of actions is then post-serializable if:

1. the union of their preconditions is consistent, which is to say that

   $$f \in P \rightarrow \neg f \notin P$$

   where $P := P_{a0} \bigcup \ldots \bigcup P_{an}$ for all $a_i \in A$;

2. the union of their effects is consistent; and

3. A_G is acyclic.

After plan generation the actions are serialized in a post-processing phase by iteratively removing all nodes from A_G that have in-degree 0. This process creates a ∀-step plan in which each action set is applied in parallel in the order in which they are removed from A_G.

Rintanen et al. [89] applied post-serialization to Planning as satisfiability, calling it *1-Linearization Semantics* and found the encoding to be very efficient, later reformalising the step semantics as $\exists$-step semantics [90]. $\exists$-step semantics uses a *disabling graph* rather than a preconditions-effects graph. While related, the disabling graph often has far fewer edges and much smaller strongly connected components. This property leads to a much more efficient encoding of the step-semantic constraints. A disabling graph for a set of actions A is the graph A_D in which each action is represented by a node and if an edge $\langle a_i, a_j \rangle$ exists then:

1. the union of the preconditions of a_i and a_j is consistent;
2. there exists a reachable state $s \models P$, where $P := P_{ai} \bigcup P_{aj}$;
3. the union of the effects of a_i and a_j is consistent; and
4. the effects of a_i are inconsistent with the preconditions of a_j.

Computing minimal disabling graphs is NP-hard due to the consistency checks and PSPACE-hard due to the reachability test. Rintanten et al. compute non-minimal disabling graphs by approximating these conditions.

The actions of each action set corresponding to a strongly connected component of the disabling graph are then given some arbitrary order, $a_0, \ldots a_n$. Mutual exclusion constraints, similar to those in $\forall$-step semantics, are included for two actions a_i and a_j if they are mutually exclusive and $i < j$. This ensures that there exists a valid serialization of the actions within each parallel step corresponding to the arbitrary ordering of the strongly connected components. This also means that the encoding only models a subset of the number of parallel plans that accord to the semantics. However, the approach does not lose completeness as fewer actions are applicable in parallel if the graph is not minimal. The resulting encoding allows for greater parallelism than $\forall$-step semantics and also a smaller number of mutual exclusion constraints.

Wehrle and Rintanen [96] extended this idea to further relax the parallelism constraints. Previously actions could only be applied in parallel if all of their preconditions were satisfied at the beginning of the time-step. Wehrle and Rintanen allow actions to be applied in parallel if they are instead enabled by some action executed in the same time-step. In order to achieve this Wehrle and Rintanen introduced alternate precondition and parallelism axioms, which are described below.

One necessary condition to the relaxed $\exists$-step semantics is that two actions applied in parallel cannot conflict. Consider two actions a_i and a_j. If there are literals $m_i \in P_{a_i}$ and $m_j \in P_{a_j}$ such that:

1. there is no reachable state $s \models m_i \wedge m_j$; and
2. there is no action a such that:
    - $m_j \in E_a$,
    - E_a is consistent with P_{a_j}, and
    - E_a,E_{a_i} and E_a,E_{a_j} are consistent,

then a_i and a_j conflict in the ordering $a_i < a_j$. If they conflict in both orderings, then they conflict.

In order to capture the subsets of actions that could be applied in parallel a new graph is used, the *disabling-enabling graph*; a generalised version of the disabling graph. The disabling-enabling graph A_{DE} for a set of actions A is the graph $\langle A, E\rangle$. There exists an edge $\langle a_i, a_j\rangle$ if:

1. a_i and a_j do not conflict;
2. there is a literal l such that $l \in E_{a_i}$ and $l \in P_{a_j}$ (enabling), or $\neg l \in E_{a_j}$ and $l \in P_{a_i}$ (disabling); and
3. there is a reachable state $s \models E_{a_i} \bigcup E_{a_j}$.

As with the disabling graph, the disabling-enabling graph calculated is non-minimal, but computed in polynomial time. Any set of actions which form a subset of a strongly connected component of the disabling-enabling graph cannot be applied in parallel. Each strongly connected component S is given a fixed ordering, $a_0 < \ldots < a_n$. The successor set of an action in a strongly connected component is defined as:

$$succ(S, a_i) := \{a_j \in S, j > i\}.$$

Consider an example precondition constraint in a $\forall$-step semantic encoding:

$$a \rightarrow p, \forall p \in P_a.$$

With this constraint, applying action a in state s ensures that $s \models P_a$. Instead the relaxed semantics allow the preconditions of a to be achieved by actions applied in parallel with a. These actions are called *enabling* actions. Possible enabling actions are identified for each action per precondition. An action a_i is an enabling action for a with respect to precondition p if

- $a_i \neq a$;
- a_i and a do not conflict in the ordering $a_i < a$;
- a_i has consistent effects with a; and
- $p \in E_{a_i}$.

All the enabling actions for a with respect to p are represented by $en(a, p)$. The precondition constraint becomes:

$$a \rightarrow (p \vee \bigvee_{a_i \in \{en(a,p) \setminus succ(S,a)\}} a_i)), \forall p \in P_a$$

where S is the unique strongly connected component in which a occurs.

With this definition of the precondition constraints it is sufficient for the parallelism constraints to ensure that no action disables another when applied in parallel. An action a_j that is disabled by action a_i, where a_i is ordered before a_j in the strongly connected component, must be applied in a later time-step.

Similar to Rintanen et al. [90], Rintanen and Wehrle achieve this with the chain-formula for each strongly connected component S:

$$\begin{gathered} a_i \rightarrow m_p^j,\, i < j,\, a_i \in D_p,\, \{a_j \in S \,|\, p \in P_{a_j}\},\, \{a_{i+1}, \ldots, a_{j-1}\} \cap D_p = \emptyset \\ \wedge \\ m_p^i \rightarrow m_p^j,\, i < j,\, a_i, a_j \in D_p,\, \{a_{i+1}, \ldots, a_{j-1}\} \cap D_p = \emptyset \\ \wedge \\ m_p^j \rightarrow \neg a_j,\, \{a_j \in S \,|\, p \in P_{a_j}\} \end{gathered}$$

where $D_p := \{a \in S | \neg p \in E_a\}$ is the set of actions in S which delete p. The auxiliary variable m_p^j is true if there is an action $a_i \in D_p$ that is applied, with $i < j$. The intuition behind this formula is that once an action that falsifies p becomes true, any action ordered afterwards and requiring p as a precondition cannot be applied. The number of auxiliary variables is linear in the number of actions.

The relaxed $\exists$-step semantics are used in the 2010 planner Madagascar, developed by Rintanen [86], representing the state-of-the-art in step semantics for state-based SAT encodings. Relaxed $\exists$-step semantics can be applied as-is to the QBF encodings presented in Chapter 3.

Split Representation of Actions

Originally proposed by Kautz and Selman [55] the idea of operator splitting was implemented in the planner MEDIC by Ernst et al. [26]. This planner, following the original ideas of Kautz and Selman, operated in a linear encoding, not a parallel one. Robinson et al. noted that this suffered from size blowup and proposed an encoding that used the split operator representations of MEDIC in a semi-parallel setting [91]. This encoding disallowed some valid parallel applications of actions that the action representation was not expressive enough to capture.

This work was extended by Robinson et al. from the approximate to the optimal parallel setting with the planner SOLE [92]. The motivation behind this encoding was to treat the size problems inherent in SAT encodings of Planning problems using a smarter action representation.

In previous work operators were split based on their parameters, for example the operator $(\textit{stack}\ ?b1\ ?b2)$ would become $(\textit{stack}[1]\ ?b)$ and $(\textit{stack}[2]\ ?b)$. Robinson et al. called this *simple splitting* and outlined two benefits and one drawback of the representation. The first advantage is one of size. Once grounded, the number of variables in a direct encoding would be reduced from $|B|^2$ to $2|B|$, where $|B|$ is the number of blocks. The second advantage is the reduction in the number of constraints required to represent these actions, such as frame axioms, precondition, effect and mutual exclusion constraints. For example the constraint:

$$(\textit{unstack}\ ?b\ B)_i \rightarrow (\textit{clear}\ B)_{i+1}$$

would be replicated for each block $b \in \textit{Dom}$. A split action representation instead requires only a single variable $(\textit{unstack}[2]\ B)_i$ for each grounding of $(\textit{unstack}\ ?b\ B)_i$ and a single constraint of the form:

$$(\textit{unstack}[2]\ B)_i \rightarrow (\textit{clear}\ B)_{i+1}.$$

The main disadvantage to simple splitting is interference between actions in a parallel plan encoding. For example, using a simply-split action representation the actions $(\mathit{unstack}\,A\,B)_i$ and $(\mathit{unstack}\,C\,D)_i$ executed in parallel would involve four split actions:

$$(\mathit{unstack}[1]\,A)_i,\ (\mathit{unstack}[1]\,C)_i,\ (\mathit{unstack}[2]\,B)_i \text{ and } (\mathit{unstack}[2]\,D)_i.$$

As the actions $(\mathit{unstack}\,C\,B)_i$ and $(\mathit{unstack}\,A\,D)_i$ involve the same split actions, it is ambiguous which split actions are paired.

Robinson et al. combat this deficiency with a *precisely split* representation. To begin with they reformulate the operators of the Planning problem into sets of pre and post-conditions. Operators are then represented as a set of composite conditions. For example, the operator O:

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

is represented in the form $\langle O, C\rangle$, where C is a set of condition composites:

$$\begin{aligned}&\langle(\mathit{stack}\,?x\,?y),\\ &\quad\{PRE(\mathit{holding}\,?x),\ DEL(\mathit{holding}\,?x)\},\\ &\quad\{PRE(\mathit{clear}\,?y),\ DEL(\mathit{clear}\,?y)\},\\ &\quad\{ADD(\mathit{handempty})\},\\ &\quad\{ADD(\mathit{on}\,?x\,?y)\},\\ &\quad\{ADD(\mathit{clear}\,?x)\}\rangle.\end{aligned}$$

This representation is then grounded.

The precisely split representation has the important property that any set of actions applied in parallel is represented by a unique conjunct of the condition variables, except in the case of redundant actions. An action a is said to be redundant if there is a parallel execution of action set A and any state reached from the application of A is indistinguishable to those reached by $A\backslash a$. Therefore, precise splitting can be used to encode step-optimal parallel Planning.

The encoding used by SOLE is a Graphplan-based encoding. It involves a variable for each ground condition, fluent and some auxiliary variables: auxiliary conditions used in mutex constraints and condition copies used in parallel step semantic constraints. The notation OC^a will be used for ground condition C of operator O and action a and $\hat{OC}^a$ for auxiliary condition variables of ground condition OC corresponding to action a. $\mathcal{C}$ represents the set of all ground conditions.

The first step is the generation of *dependency trees.* A dependency tree is generated from each condition that includes an ADD term with the ground condition as the root node n_0. $\mathit{parent}(n)$ is used for the parent of n, $\mathit{prefix}(n)$ for the root path of n and $\mathit{children}(n)$ for the children of n. The possible children of a node are the ground conditions with which it co-occurs in a conjunct

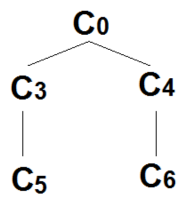


Figure 2.3: The restricted dependency tree of the ground condition $\langle (\textit{unstack}\,?x\,?y), \{ADD(\textit{holding}\,A)\}\rangle$.

representing a whole action. The actual children are those that only instantiate and exhaust the instantiations of one ungrounded condition. For example the constraint C_0, $\langle (\textit{unstack}\,?x\,?y), \{ADD(\textit{holding}\,A)\}\rangle$, has the associated ground conditions:

$$\begin{aligned}
C_1 &:= \{PRE(\textit{clear}\,A),\, DEL(\textit{clear}\,A)\}\\
C_2 &:= \{PRE(\textit{handemtpy}),\, DEL(\textit{handempty})\}\\
C_3 &:= \{PRE(\textit{on}\,A\,B),\, DEL(\textit{on}\,A\,B)\}\\
C_4 &:= \{PRE(\textit{on}\,A\,C),\, DEL(\textit{on}\,A\,C)\}\\
C_5 &:= \{ADD(\textit{clear}\,B)\}\\
C_6 &:= \{ADD(\textit{clear}\,C)\}.
\end{aligned}$$

The conditions C_1 and C_2 are omitted from the tree as they are present in any conjunction that forms the instantiation of the unstack operator. Their dependency is instead represented by a straight implication $C_0 \rightarrow C_1 \wedge C_2$. The dependency tree of C_0 is represented in Figure 2.3. Constraint (10) in Figure 2.4 then ensures that the whole action is executed.

The constraints involved in the encoding are described by Figure 2.4. Constraints (1) and (2) assert the initial state and goal. Constraints (3), (4), and (5) assert action preconditions and effects, while constraint (6) is an example of an explanatory frame axiom. Fluent mutual exclusion is asserted as constraint (7). Fluent mutual exclusion in combination with constraints (3), (4), and (5) deal with most action mutual exclusion relations leaving only those of the form: $O_x C_i^a \rightarrow p_i$ and $O_y C_i^a \rightarrow \neg p_{i+1}$. This situation is dealt with in two ways: when $O_x \neq O_y$ by constraint (8) and when $O_x = O_y$ by constraint (9).

Constraint (8) asserts that the ground condition belonging to one of the mutually exclusive actions must be false.

Constraint (9) involves the auxiliary condition variables and constrains the case when there are mutually exclusive actions a and b instantiated from the operator O. For example the split actions $(\textit{stack}\,A\,B)$ and $(\textit{stack}\,A\,C)$ have a mutually exclusive relationship involving the ground constraint

$$\langle (\textit{stack}\,?x\,?y), \{PRE(\textit{holding}\,A),\, DEL(\textit{holding}\,A)\}\rangle$$

not covered by the fluent mutual exclusivity constraints. In this case auxiliary condition variables $\hat{OC}^a$ and $\hat{OC}^b$ are instantiated and made mutually exclusive. In the grounding support constraints (10) $a \rightarrow \hat{OC}^a$ and $b \rightarrow \hat{OC}^b$ are enforced.

Constraint (10) ensures that whole actions are executed, rather than just individual conditions.

1. $(p \in I) \leftrightarrow p_0$
2. $(p \in G) \rightarrow p_n$
3. $\bigwedge_{OC}^{PRE(p)\in\mathcal{C}}(OC_i \rightarrow p_i)$, $i = 1, \ldots, n$
4. $\bigwedge_{OC}^{ADD(p)\in\mathcal{C}}(OC_i \rightarrow p_{i+1})$, $i = 1, \ldots, n-1$
5. $\bigwedge_{OC}^{DEL(p)\in\mathcal{C}}(OC_i \rightarrow \neg p_{i+1})$, $i = 1, \ldots, n-1$
6. $p_{i+1} \rightarrow (p_i \vee \bigvee_{OC \in ach(p)} OC_i)$, where $ach(p)$ is the set of all ground conditions OC such that $ADD(p) \in OC$, $i = 2 \ldots n$
7. $\neg p_i \vee \neg p'_i$ for each pair of fluents p, p' that are mutually exclusive in layer i of the plangraph, $i = 1 \ldots n$
8. $\neg O_1C_i^a \vee \neg O_2C_i^b$ for each pair of ground conditions $O_1C^a \neq O_2C^b$ such that there exists fluent p where $PRE(p) \in O_1C^a$ and $DEL(p) \in O_2C^b$, $i = 1 \ldots n$
9. $(\neg O_1\hat{C^a}_i \vee \neg O_1\hat{C^b}_i)$ for each ground condition O_1C such that there exist two mutually exclusive actions a and b, $i = 1 \ldots n$
10. $(\bigwedge_{\substack{n_x \in prefix(n)\bigcup\{n\},\\ |children(parent(nx))|>1}} OC_i^{nx}) \wedge OC_i^{no} \rightarrow \bigvee_{n_y \in children(n)} OC_i^{ny}$, $i = 1 \ldots n$

Figure 2.4: Constraints for the split action encoding used by SOLE.

The constraints provided by the dependency tree and (10) are overly restrictive, not allowing as much parallelism as the semantics dictate. In order to relax this restriction it is necessary to deal with the case where two actions instantiated from the same operator that are not mutually exclusive, but share a common ground condition, can be applied in parallel. This is accomplished by the addition of condition copy variables.

Robinson et al. found that SOLE dominated SATPLAN'06 in terms of compactness of encoding. The precisely split operator representation can be applied directly to the QBF encodings presented in Chapter 3. The QBF encoding presented in Chapter 4 already uses the simply-split operator representation, and the precisely split operator representation could be used instead; however, since the encoding is not grounded it is not clear whether a more compact encoding would be obtained.

SAS$^+$-based encodings

The simple action structures (SAS$^+$) formalism was described by Bäckström [2] as a modification of the traditional STRIPS formalism of Fikes and Nilsson [29]. SAS$^+$ uses multi-valued state variables rather than propositional ones and a prevail condition in addition of pre- and post-conditions. The formalism is compact, with enough structural information to garner interest for use in deriving heuristics [47], landmarks [77] and stronger mutual exclusions [16].

Huang et al. developed a translation from Planning to SAT using the SAS$^+$ formalism which they call the SASE encoding [50, 51]. While previous translations encoded actions and fluents as variables in the SAT formula, either directly or split, the new encoding contains variables representing transitions in the multi-valued variables of the underlying SAS$^+$ formulation of the problem.

A Planning task in the SAS$^+$ formalism is defined as the 4-tuple $\langle \mathcal{X}, \mathcal{A}, I, G \rangle$ in which:

- $\mathcal{X} := \{x_1, \ldots, x_n\}$ is a set of state variables with associated domains $Dom(x_i)$;
- $\mathcal{A}$ is a set of actions;
- I is a full set of assignments to $\mathcal{X}$ representing the initial state; and
- G is a partial set of assignments to $\mathcal{X}$ specifying the goal.

A transition for some given variable x_i is a re-assignment of x_i. The re-assignment from $x_i = f$ to $x_i = g$, $f, g \in Dom(x_i)$, will be written as $\delta^{x_i}_{f \to g}$. The state reached from applying transition δ in state s is denoted s^δ. Transitions are one of three types: *regular*, *prevailing*, and *mechanical*:

- A regular transition $\delta^x_{f \to g}$ is applicable to a state s iff $s(x) = f$ and $s^\delta(x) = g$.
- A prevailing transition $\delta^x_{f \to f}$ is applicable to a state s iff $s(x) = f$ and $s^\delta = s$.
- A mechanical transition $\delta^x_{* \to g}$ is applicable in any state s and $s^\delta(x) = g$.

1. $\bigvee \delta^x_{f \to g,1}$ such that $I \models (x = f)$
2. $\bigvee \delta^x_{f \to g,n}$ such that $G \models (x = g)$
3. $\delta^x_{f \to g,i+1} \to \bigvee \delta^x_{f' \to f,i}$ for all $i = 1 \dots n$
4. $\neg\delta_1 \vee \neg\delta_2$, for all pairs of mutually exclusive transitions, δ_1, δ_2, and for $i = 1 \dots n$
5. $\bigvee \delta^x_i$ for all $x \in \mathcal{X}$ and $i = 1 \dots n$
6. $a_i \to \bigwedge \delta_i$ for $i = 1 \dots n$ and $\delta \in a$
7. $\delta_i \to \bigvee a_i$ for all $i = 1 \dots n$ and $a \,|\, \delta_i \in a$
8. $\neg a_1 \vee \neg a_2$, $\forall a_1, a_2 \in \mathcal{A}$ such that a_1 and a_2 are mutually exclusive, for $i = 1 \dots n$

Figure 2.5: Constraints for the SAS$^+$-based encoding used by SASE.

Each type is treated differently in the encoding. An action $a \in \mathcal{A}$ is comprised of a set of transitions. Huang et al. describe transitions as the atomic elements of state transitions and note that there are usually far fewer actions in the domain. The SASE encoding encodes the transitions directly, a second set of constraints forming a second layer of logic.

The SASE encoding includes a variable at every time-step for each transition $\delta \in T$ and for each action $a \in \mathcal{A}$. There are eight classes of constraint upon these variables that make up τ and σ. These are shown in Figure 2.5.

In these constraints $\delta^x_{f \to g,i}$, or simply δ_i, refer to the variable representing the transition in time-step i. Constraints (1) and (2) specify the initial and goal states. Constraint (3) constrains the transitions to follow a valid plan by ensuring that changes to the assignment of a variable correspond to an applied transition in the domain. Two transitions are mutually exclusive if they act upon the same variable; neither are mechanical; both are mechanical; or exactly one is mechanical and they transit to the same assignment. With this definition, constraints (4) and (5) ensure that exactly one transition is applied to each state variable. Constraints (6) and (7) form the second layer of logic, linking the transitions to actions, while constraint (8) enforces action mutual exclusions.

A number of methods for reducing the size of the encoding were proposed by Huang et al. [50], taking into account the clique structure of mutual exclusions among transitions, subsumption of transition sets between actions, and unary actions with only one transition.

Huang et al. [51] found that the SASE encoding was more efficient than SATPLAN'06 in terms of both memory and time to solve. The VSIDS heuristic [70], used in most existing SAT algorithms, selects variables based on the frequency with which they it appear in clauses. This value changes as new learned clauses are introduced and previous scores are scaled down. Huang et al. show through experimentation that the transition variables are scored much higher than action variables and chosen more often for branching, perhaps indicating that the transition variables provide more powerful propagation, explaining the improved solution time.

The SASE encoding bears some similarities to SOLE in the way in which actions are broken down into elements which may belong to more than one instantiating action. The approach of SASE is directly applicable to the QBF encodings in Chapter 3, in much the same way as the approach of SOLE. Both encodings specify a new set of variables and constraints τ and σ which are not specific to SAT.

More interesting are the possibilities for combining the ideas of SASE and the QBF encoding in Chapter 4. The encoding used by SASE could be transformed into a partially grounded QBF in the same way that the encoding used by SATPLAN'06 corresponds to the encoding described in Chapter 4. The SAS$^+$ formalism is a natural target for ungrounded representations and would lead to a more intuitive encoding. This has been left for future work.

2.1.2 Embedding Planning-Specific Knowledge

Initial analysis on the Planning problem can provide additional constraints to be embedded in the formula. Making the problem more constrained in this way can lead to better solve times, even with the overhead of the initial analysis. Two approaches are discussed here, symmetry in Planning and long-distance mutual exclusions. Both have been explored in the Planning as SAT setting. The former extends σ, the state constraints, and can be applied in the same way to any of the QBF encodings presented here. The latter involves constraints between time-steps and therefore cannot be applied directly to the encodings presented in Chapter 3. In general any constraints used in Planning as SAT that extend σ or τ can be applied to any QBF encoding in the same way. Constraints that involve non-adjacent states may require additional variables to link involved states that lie in different contexts of the QBF encodings in Chapter 3. This will limit, or remove, the benefit obtained from applying the additional constraints in the first place. The QBF encoding in Chapter 4 does not share this limitation as the states are distinctly described in the same way as in a SAT representation.

Symmetry in Planning as Satisfiability

Planning problems often have a high degree of underlying symmetry. For example, consider the domain in Figure 2.6 and problem in Figure 2.7, which will be used as an example in this section. The problem, which involves a robot moving balls from one room to another, has a high degree of symmetry. The balls are identical, as are the two grippers. This problem is easy to solve, but if the symmetries are not recognised proving optimality becomes very difficult. Exploring every permutation of ball ordering and gripper assignments is computationally expensive.

Fox and Long [31, 33] showed that the symmetries could be discovered and exploited during the search process of a Graphplan-style Planning system. Rintanen [81] proposed an improvement for propositional logic-based systems in which more symmetry is discovered, introducing symmetry-breaking constraints with the intention to break symmetry over symmetric transitions, not just states or single actions.

Rintanen described a state-based encoding with classical frame axioms as a basis upon which to apply the new symmetry-breaking constraints.

```
(define (domain gripper-typed)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robby ?r - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
                (carry ?o - ball ?g - gripper))
   (:action move
       :parameters  (?from ?to - room)
       :precondition (at-robby ?from)
       :effect (and
                (at-robby ?to)
                (not (at-robby ?from))))
   (:action pick
       :parameters (
                ?obj - ball
                ?room - room
                ?gripper - gripper)
       :precondition  (and  (at ?obj ?room)
                            (at-robby ?room)
                            (free ?gripper))
       :effect (and (carry ?obj ?gripper)
                    (not (at ?obj ?room))
                    (not (free ?gripper))))
   (:action drop
       :parameters  (
                ?obj - ball
                ?room - room
                ?gripper - gripper)
       :precondition  (and
                (carry ?obj ?gripper)
                (at-robby ?room))
       :effect (and (at ?obj ?room)
                    (free ?gripper)
                    (not (carry ?obj ?gripper)))))
```

Figure 2.6: Gripper domain, adapted from the AIPS-98 Planning competition.

```
(define (problem gripper2)
(:domain gripper-typed)
(:objects roomA roomB - room ball1 ball2 - ball)
(:init
      (at-robby roomA)
      (free left)
      (free right)
      (at ball1 roomA)
      (at ball2 roomA))
(:goal (and (at ball1 roomB) (at ball2 roomB))))
```

Figure 2.7: A simple instance of gripper.

First symmetries are identified between objects by looking at the operators in the domain description and goal definition. Only identical objects which do not appear individually in operators and have the same goal state (if they appear in a goal at all) are considered to be symmetrical. Once this is done an ordering is placed upon the actions involving the symmetrical objects. For example, in the *gripper* domain and problem in Figures 2.6 and 2.7, *ball*1 and *ball*2 are symmetrical, as are $left$ and *right*. Let the imposed ordering be *ball*1 $<$ *ball*2 and $left < right$, then the actions instantiated from the operator *pick* are ordered:

$$\begin{gathered}
(pick\, ball1\, roomA\, left) < (pick\, ball1\, roomA\, right) \\
(pick\, ball2\, roomA\, left) < (pick\, ball1\, roomA\, right) \\
(pick\, ball1\, roomA\, left) < (pick\, ball2\, roomA\, left) \\
(pick\, ball1\, roomA\, right) < (pick\, ball2\, roomA\, right).
\end{gathered}$$

A constraint can be introduced to break the symmetry upon these objects by first checking whether the objects are in the same state, with respect to their symmetry, and then only applying the first symmetrical action. This takes the form[1]:

$$((v_1 \leftrightarrow v_1') \wedge \ldots \wedge (v_n \leftrightarrow v_n')) \rightarrow (a_1 \leftarrow a_2)$$

where $v_1, \ldots, v_n$ are the state variables involving the first object; v_i, v_i' are pairs of state variables obtained by replacing occurrences of the first object with the second; and a_1 and a_2 are the action pairs symmetric with respect to the objects, with $a_1 < a_2$. In the case of the example of the pair of actions in the fourth row the constraint is:

$$\begin{gathered}
\left(\begin{gathered}
(at\, ball1\, roomA) \leftrightarrow (at\, ball2\, roomA) \\
\wedge(at\, ball1\, roomB) \leftrightarrow (at\, ball2\, roomB) \\
\wedge(carry\, ball1\, left) \leftrightarrow (carry\, ball2\, left) \\
\wedge(carry\, ball1\, right) \leftrightarrow (carry\, ball2\, right)
\end{gathered}\right) \\
\rightarrow \\
(pick\, ball1\, roomA\, right) \leftarrow (pick\, ball2\, roomA\, right).
\end{gathered}$$

[1]This constraint is presented as described in Rintanen 2003 [81]. The implication ($a_1 \leftarrow a_2$) seems incorrect, but should be read a_1 is applied *instead of* a_2. Similarly for (*pick ball*1 *roomA* $left$) and (*pick ball*1 *roomA right*) in the example constraint, the left arrow is used to mean *instead of*.

Rintanen further improved this approach by taking into consideration parallel application of actions. For example if the action $(pick\,ball1\,roomA\,left)$ were applied it should be possible to apply in parallel $(pick\,ball2\,roomA\,right)$, under any of the parallel step-semantics described earlier. However, due to the ordering constraints $(pick\,ball1\,roomA\,right)$ is to be applied instead, which is mutually exclusive with $(pick\,ball1\,roomA\,left)$. This is handled by using modified constraints of the form:

$$\begin{array}{l}((v_1 \leftrightarrow v_1') \wedge \ldots \wedge (v_n \leftrightarrow v_n') \\ \qquad \neg\hat{a_1} \wedge \ldots \wedge \neg\hat{a_m}) \rightarrow (a_1 \leftarrow a_2)\end{array}$$

where $\hat{a_1}, \ldots, \hat{a_m}$ are the actions ordered before a_1 that are mutually exclusive with a_1. The modified example constraint becomes:

$$\begin{array}{l}\left(\begin{array}{c}(at\,ball1\,roomA) \leftrightarrow (at\,ball2\,roomA) \\ \wedge(at\,ball1\,roomB) \leftrightarrow (at\,ball2\,roomB) \\ \wedge(carry\,ball1\,left) \leftrightarrow (carry\,ball2\,left) \\ \wedge(carry\,ball1\,right) \leftrightarrow (carry\,ball2\,right) \\ \wedge\neg(pick\,ball1\,roomA\,left)\end{array}\right) \\ \rightarrow \\ (pick\,ball1\,roomA\,right) \leftarrow (pick\,ball2\,roomA\,right).\end{array}$$

These constraints, quadratic in the number of actions, can greatly increase the size of the resultant encoding. Experimentation on the *gripper* domain showed that they improve both time to solve and the scalability of the approach. Rintanen was able to show that using the constraints, proof of optimality was possible for up to 20 balls in a *gripper* problem, and without the constraints optimality could not be proved for even 8 balls.

As mentioned this approach is an extension of the state-constraints (σ) and can be applied directly to the QBF encodings presented in this work.

Long-Distance Mutual Exclusion

Chen et al. [16, 17] proposed a generalisation of mutual exclusion relations to capture constraints over fluents and actions across multiple time-steps. These constraints are derived from the SAS$^+$ representation [2] of the Planning problem and can be easily encoded in propositional logic. Chen et al. describe two types of long-distance mutual exclusion (*londex*) constraint, londex$_1$ and londex$_m$. The latter is an extension of the londex$_1$ constraints made stronger by the inclusion of causal dependencies.

First londex constraints between fluents are discovered; londex constraints between actions are then derived from them. Suppose we have mutually exclusive fluents f and f' and states $s_i \models f$ and $s_{i'} \models f'$. Intuitively a londex constraint models the lower bound on the number of time-steps that must occur between the two states. That is, a lower bound on $i' - i$. These lower bounds are computed from the *domain transition graphs* of the problem and are denoted $d(f, f')$.

A domain transition graph (DTG) for a SAS$^+$ variable $x \in \mathcal{X}$ is the directed graph G_x with nodes $Dom(x)$. An edge exists between nodes (f, g) if there is a transition between the two assignments, $\delta^x_{f \rightarrow g}$.

A londex$_1$ constraint models a long-distance mutual exclusion from a single DTG. Given two fluents f and f' which both correspond to assignments to the same variable x, and therefore correspond to unique nodes in the same DTG, the shortest path between them is computed. $d(f, f')$ is the length of the shortest path. If f is true at time-step i and f' at i' there exists no plan for which:

$$0 \leq i' - i < d(f, f').$$

The following constraint set is added to the formula, in addition to the usual mutual exclusion constraints, if fluent variables are used:

$$f_i \rightarrow \neg f'_j$$

for all f' mutually exclusive with f and $j = i + 1, \ldots, (d(f, f') - 1)$.

Action londex constraints are derived from fluent londex constraints. Suppose for mutually exclusive fluents f and f', $d(f, f') = r$. We have two fluents a and b associated with f and f' respectively. The action londex constraints are included in the encoding in the same way as are the fluent londex constraints using the following rules:

- if $f \in E_a$ and $f' \in E_b$ then $d(a, b) \geq r$;
- if $f \in E_a$ and $f' \in P_b$ then $d(a, b) \geq r + 1$;
- if $f \in P_a$ and $f' \in E_b$ then $d(a, b) \geq r - 1$;
- if $f \in P_a$ and $f' \in P_b$ then $d(a, b) \geq r$.

Since action londex constraints are derived from fluent londex constraints the method for finding londex$_m$ constraints is only stronger for fluents. The action londex constraints are derived from the stronger fluent constraints using the same rules. Longdex$_m$ constraints improve upon londex$_1$ constraints by taking into account the dependencies between DTGs. Suppose a transition in DTG$_x$, $\delta^x_{f \rightarrow g}$, belongs to action a with precondition $f' \in P_a$, where f' corresponds to the assignment $x' = h$. Then h appears as a node in DTG$_{x'}$. We say that DTG$_x$ depends upon DTG$'_x$.

Chen et al. [17] construct a causal dependency graph between DTGs in which a directed edge exists between DTGs (v, v') if and only if v depends on v'. They also construct an *invariant connectivity graph* (ICG). This graph corresponds to invariants [32] and can be thought of as a partially grounded causal dependency graph. Two methods are described for obtaining strong londex constraints using the ICG [16], a technique based on shared preconditioning and bridge analysis.

The idea of shared preconditions stems from the consideration that some transitions in one DTG always require transitions in another DTG upon which it is dependent. For example consider two fluents f and f' in DTG G$_x$ for which the minimum distance is $d_1(f, f')$ using londex$_1$. Suppose we restrict ourselves to looking only at the shortest path between these fluents that passes through fluents v and w, $\xi := (f, v, \ldots, w, g)$. If $\delta_{f \rightarrow v}$ has the precondition p and $\delta_{w \rightarrow f'}$ has the precondition p' with both $p, p' \in \mathrm{G}_{x'}$ and $d(p, p') \geq d(f, f')$ then the minimum distance between f and f' passing through v and w can be updated with $d(p, p') + 1$. This procedure can be repeated for each pair of v and w such that there exist transitions $\delta_{f \rightarrow v}$ and $\delta_{w \rightarrow f'}$ and both transitions have a precondition in the same DTG upon which G$_x$ is dependent. The minimum distance $d(f, f')$ between f and f', without other constraints upon the path, is then the minimum distance computed over all combinations of v and w.

This procedure can be completed recursively, the minimum distance between $d(p, p')$ being updated in the same way by looking at the DTGs upon which $G_{x'}$ is dependent. Chen et al. break cycles in the ICG by constructing a spanning tree using as a root node of the DTG in which the relevent fluents reside. The tighter bound is denoted $\Upsilon(f, f')$. It is important to note that the lower bound upon the distance between two fluents is for parallel plans with $\forall$-step semantics.

Bridge analysis provides tighter bounds on $\Upsilon(f, f')$ in the case where there are no fluent pairs v and w, successors and predecessors of f and f' respectively, that share common preconditions. In this case the possible paths between f and f' are analysed to find a *bridge pair*.

A pair of fluents x and y are called a bridge pair of v and w if any path between v and w through the DTG must pass through x and y in order. The path between x and y is then called a bridge. If $d(x, y)$ is improved to $\Upsilon(x, y)$ through shared preconditions then any pair of fluents v, w for which x, y is a bridge pair can be updated:

$$\Upsilon(v, w) = d(v, x) + \Upsilon(x, y) + d(y, w).$$

DTGs are usually sparse and the existence of bridge pairs common.

Mutual exclusion relations form the majority of clauses in Graphplan and state-based encodings and adding londex constraints increases the number of mutual exclusion clauses. Chen et al. noted that these binary constraints are responsible for the majority of the propagation during search and such an increase in them should lead to faster solution times. They discovered through experimentation that this was indeed the case. Chen et al. incorporated londex constraints into both SATPLAN'04 and SATPLAN'06. The former, using an action-only encoding, was called MAXPLAN [17] and came first in the 5th IPC in 2006 [36] (jointly with SATPLAN'06).

The disadvantage of including londex constraints is that the increased number of clauses can make the encoding untenably large. Chen et al. [16] devised an on-the-fly method of using the londex constraints in a modified version of MINISAT [25] as non clausal constraints.

Longdex constraints are unusual in that they are not easily applied in a clausal form to all of the QBF encodings presented in this work. Specifically the encodings in Chapter 3 do not explicitly represent every time-step with unique variables. Due to this it is not possible to simply represent a constraint between elements in separate time-steps without the inclusion of more variables in an earlier quantifier set. This will negatively impact the effectiveness of this technique in the QBF encoding.

2.1.3 Solver Modifications and Search Algorithms

The techniques presented in this section fall into two categories: modifications to a SAT solver; modifications to the solving algorithm.

The former category, modifications to the SAT solver, is based upon the premise: once translated into a propositional formula the underlying structure of the original Planning problem persists, but is largely ignored. Modifications to SAT solvers to take advantage of Planning-specific knowledge and techniques can provide huge improvements in time to solve. This comes with the disadvantage that the SAT-based Planning system no longer improves alongside improvements in SAT solving, as the latest SAT solvers must first be modified and cannot just be run “off the shelf”. The solver modifications can be applied directly to QBF solvers as long as there is sufficient similarity between the QBF encodings presented in this paper and SAT-based encodings underlying these techniques, and there is sufficient similarity between the solving algorithm of the QBF and SAT solvers. The modifications described here are all applicable to the encodings presented in Chapters 3 and 4. In the work described below, modified SAT solvers rely on replacing the heuristic in some variant of the Davis–Putnam [20] search algorithm. The new heuristic, used to pick variables upon which to branch, can be used in a number of QBF solvers; in particular QUBE and DepQBF.

The latter category, modifications to the solving algorithm, can be seen as changes to the top-level algorithm of the Planning system. These change the way in which the encodings are generated and passed to the solver. These techniques are all possible with the encodings presented in this work. In particular the work of Ray and Ginsberg [76] has already been implemented using the Compact Tree Encoding described in Chapter 3 with the solver QUBE.

Planning-specific Heuristics for SAT Solvers

Rintanen [88] wrote that the performance gap between SAT-based planners and the best planners overall has been percieved to be prohibitively wide. However, with the the introduction of Planning-specific heuristics to SAT solving the gap disappears. His Planning system Madagascar [86] uses this approach to achieve competitive performance.

This idea is not new; Giunchiglia et al. [37] implemented a modified Planning system, generating encodings with Medic [26] and solving them with Tableau*, a modified version of Tableau [19]. Their modifications stemmed from the consideration that the fluents at any time point are derived deterministically from the initial state and the sequence of actions applied up to that point. They use this knowledge to drastically reduce the size of the search space used by the SAT solver, limiting the branching to variables that represent actions.

Consider the standard Davis–Putnam algorithm for solving SAT problems described in Figure 2.8. Selecting a literal upon which to branch in step (8) involves a heuristic and is the only non-deterministic step. Generic SAT solvers commonly use the conflict-directed clause learning (CDCL) algorithm, an extension of the DPLL algorithm introduced by the SAT solver Grasp [65]. As with the algorithm in Figure 2.8 the choice of branching literal can be arbitrary, and therefore based upon a heuristic. Modern SAT solvers commonly use the VSIDS heuristic [70].

Giunchiglia et al. experimented with a number of encodings including: parallel and sequential semantics; regular, simply-split and bitwise action representations; and explanatory and classical frame axioms. If we assume a regular action representation and explanatory frame axioms, the SAT encoding would include clauses of the following form:

$$\begin{gathered} a_i \rightarrow f_i,\ \forall f \in P_a \\ \wedge \\ a_i \rightarrow f_{i+1},\ \forall f \in E_a \end{gathered}$$

```
procedure DP($\Pi, M$)                    ▷ Set of clauses $\Pi$ with partial model $M$
    if $\Pi = \emptyset$ then
        return true;
    else if $\emptyset \in \Pi$ then
        return false;
    end if
    UNIT-PROPOGATE($\Pi, M$);
    $L :=$ a literal such that $(L \in \Pi) \vee (\neg L \in \Pi)$;
    return DP($\Pi \cup \{L\}$) $\vee$ DP($\Pi \cup \{\neg L\}$);
end procedure

function UNIT-PROPAGATE($\Pi, M$)
    while there is a unit clause $\{L\} \in \Pi$ do
        $M := M \cup \{L\}$;
        for every clause $C \in \Pi$ do
            if $L \in C$ then $\Pi := \Pi \backslash \{C\}$;
            else if $\neg L \in C$ then $C := C \backslash \{\neg L\}$;
        end for
    end while
end function
```

Figure 2.8: The Davis–Putnam algorithm for SAT.

for each action a; and

$$f_i \rightarrow \bigvee_{a \in ach(f)} a_{i-1}$$

for each fluent f. Note that selecting an action a and assigning it true leaves a set of unit clauses that UNIT-PROPAGATE will discover, making assignments to all fluents $f \in E_a \cup P_a$. Choosing a fluent variable and assigning it true does not have the same effect. Giunchiglia et al. propose that the selection in step (8) of the algorithm in Figure 2.8 be limited to only the literals which represent the choice of an action. Their experimental results indicate that on some problems this simple idea can provide solving times that are up to four orders of magnitude faster.

Rintanen [84, 85] introduced a Planning-specific heuristic that does more than restrict the search space, instead completely governing the choice of literal. The new heuristic is used in the Planning system MADAGASCAR [86]. MADAGASCAR uses the $\exists$-step semantics of Rintanen et al. [90] and an alternate top-level strategy described in the next subsection as Algorithm B.

The new Planning-specific heuristic takes into consideration the state of the SAT solver, in particular the partial assignment to the variables. The heuristic bears some similarities to open condition resolution in least commitment Planning. The selection is based upon goal, or subgoal fluents that are not yet supported by an action or the initial state. In this case a subgoal refers to the preconditions of an applied action. The selection algorithm considers every goal and subgoal on each iteration, beginning with the goal literals in the final time-step. The heuristic algorithm begins by choosing a goal literal l and the earliest time-step at which:

```
procedure SUPPORT($Stack, M$)        ▷ Top-level goals and partial Model $M$
    while Stack is on-empty do
        $f_i := pop(Stack)$;
        found:= 0;
        $t = i - 1$;
        while found$\neq$ 1 and $t \geq 0$ do
            if $\exists a$ such that $(a_t \in M) \wedge (f \in E_a)$ then
                for all $f' \in P_a$ do $push(f'_t, Stack)$;
                found:= 1;
            else if $\neg f_t \in M$ then
                $a :=$ any $a \in A \,|\, (f \in E_a) \wedge (\neg a_t \notin M)$;
                return $\{a_t\}$;
            end if
            $t = t - 1$;
        end while
    end while
    return $\emptyset$;
end procedure
```

Figure 2.9: Rintanen's algorithm for finding (sub)goal support.

1. an action a is taken such that $l \in E_a$;
2. the initial state is represented and $I \models l$; or
3. the state $s \models \neg l$.

In case (1) the literals representing the preconditions of action are chosen as subgoals and their support discovered in the same way. In case (2) nothing needs to be done. In case (3) there will exist an action that can be used as support for the goal. This action is selected as the next branching literal. The algorithm is shown in more detail in Figure 2.9.

If no subgoal requiring support can be found then the current partial assignment represents a plan. A complete model for the SAT instance can be found by assigning $false$ to any remaining action variables. The fluent variables will then be determined automatically.

Rintanen [85] demonstrated that the new heuristic outperforms the VSIDS heuristic on Planning benchmarks when proofs of makespan optimality are not required. The heuristic was then generalised to actions with expressive representation, including conditional effects [87]. The new Planning-specific heuristic can be applied to any of the DPLL-based QBF solvers, such as QuBE or DepQBF. Opportunities for future research lie in the experimentation of adapting stronger Planning-based heuristics to the Boolean formula arena with analysis such as Landmarks [49, 75] and Causal Graphs [46].

Top-level Strategies for Planning as Satisfiability

The basic algorithm for Planning as propositional logic, introduced by Kautz and Selman [55] finds makespan optimal plans by encoding at iteratively larger makespans and making multiple calls to a SAT solver. The basic algorithm is

```
 1: procedure PLAN(Π)                                        ▷ Planning problem Π
 2:     $n := 0$;
 3:     $M := \emptyset$
 4:     repeat
 5:         $n := n + 1$;
 6:         $\Pi_n$ := an encoding of Π in propositional logic with makespan $n$;
 7:         $M :=$SOLVE$(\Pi_n)$;        ▷ Pass $\Pi_n$ to solver, returning the satisfying
    model if satisfiable, $\emptyset$ otherwise.
 8:     until $M \neq \emptyset$
 9:     return $M$;
10: end procedure
```

Figure 2.10: The basic algorithm for Planning as Satisfiability.

$\{5, \neg 6\}$
$\{6, 7, \neg 8\}$
$\{\neg 7, \neg 9\}$
$\{8, \neg 9\}$
$\{9, 10\}$
$\{9, \neg 10\}$
$\{9, 11\}$
$\{\neg 5\}$

Figure 2.11: A simpe SAT problem.

$1.\ \mathit{Assign}:\ \neg 5$
$\rightarrow \neg 6$
$2.\ \mathit{Assign}:\ 9$
$\rightarrow \neg 7 \rightarrow \neg \mathbf{8}$
$\rightarrow \mathbf{8}$

Figure 2.12: Solving the SAT problem leads to a conflict.

shown in Figure 2.10. Each encoding formed represents the question: *"is there a plan of length n?"*

There are certain drawbacks to this method. Firstly all learned information is thrown out between encodings. Each encoding is very similar to the last. Consider the basic state-based encoding described in Section 2.1.1 and used as an example in Chapter 1: the difference between Π_n and Π_{n+1} becomes increasingly small as the value of n increases. The two encodings differ only in that Π_{n+1} includes a single extra set of variables X with clause sets τ and σ. Information on partial assignments and learned conflicts is lost between calls to the SAT solver, even though much of the encoding remains the same.

Nabeshima et al. [71] implemented the Lemma-Reusing Planner (LRP) with the aim of avoiding this disadvantage. The idea is that *conflict clauses* learned by the CDCL algorithm [65] are extracted from one encoding and simply added to the next problem, creating tighter constraints and greater propagation, leading to a faster solution. Conflict clauses are learned when implications for setting a variable both true and false occur.

Figures 2.11 and 2.12 show this procedure in more detail using the example found in Nabeshima et al [71] with the SAT solver CHAFF [70]. Variable 9 is chosen as a decision variable at decision level 2 and this leads to an empty clause. Variable 9 is selected as the *unique implication point* (UIP). A variable x is a UIP at level l iff any implication path from the decision variable at decision level l must pass through x. Variable 9 is a UIP. Variables assigned up to and including the UIP are on the *reason side*, while the rest are on the *conflict side.*

The learned conflict clause consists of the complement of all literals that are directly connected to the reason side. In this example the learned clause

$$\{6, -9\}$$

is added to the original SAT problem. In future searches, as soon as variable 6 is assigned true, variable 9 is assigned false, pruning away the conflict that has already been encountered.

Nabeshima et al. [71] give formal correctness to the *lemma-reusability condition* between two similar SAT problems, which was proposed by Eén and Sörensson [25] and is defined as follows. For two SAT instances P and Q, if P includes a non-unit clause x, then Q contains x. This condition certainly holds for any two Graphplan or state-based encodings of Planning problems that differ only in the number of time-steps.

Learned conflicts are an important part of both DPLL-based QBF and SAT solvers. Since lemma-reusing only extracts learned constraints and adds them to the next encoding it is possible to apply this technique to the latest DPLL-based solvers for both approaches to Planning as SAT.

Ray and Ginsberg [76] approached the same problem in a more direct way. Instead of remembering information between encodings, they generate only one encoding and make a single call to a modified SAT solver, ensuring that no information is lost, while still retaining optimality. The idea is surprisingly simple; the encoding used by SATPLAN'06 [59] is augmented with a new variable at each time-step that is true iff the goal is satisfied at that level. The SAT problem file contains an extra line specifying a predetermined branching order for the SAT solver to follow. This branching order contains the new variables set to true, from the earliest time-step to the latest. This ensures that the makespan optimal solution is found, if it exists.

Effectively the Planning system is still using a ramp-up approach, iteratively deepening until it finds a solution. However, this process has been moved inside the solver and no longer requires multiple encodings. Ray and Ginsberg [76] named their Planning system CRICKET as it takes large jumps through the search space. They found that the new system could find solutions up to an order of magnitude faster than SATPLAN'06.

This approach has been applied to the Compact Tree Encoding described in Chapter 3 using QuBE7-p, a modified version of QuBE7. This technique is very natural for the Compact Tree Encoding as the number of time-steps in any given encoding are double that of the previous one. More constraints are required for traditional iterative deepening.

The second disadvantage to the algorithm in Figure 2.10 is linked to the iteratively deepening approach of the algorithm. Rintanen [82] showed that the most time-consuming encodings are those closest to an optimal solution yet still unsatisfiable. In some problems the final unsatisfiable encoding cannot be solved in any reasonable time, even though the first satisfiable encoding in the next iteration may be solved quickly.

Alternative top-level strategies which attempt to deal with this disadvantage, such as solving parallel encodings; taking different step sizes; and ramp down strategies, have all been applied to Planning as SAT [17, 82, 86, 95]. They are all applicable in the same way to Planning as QBF.

Rintanen [82] introduced two novel top-level algorithms. These procedures, called *Algorithm A* and *Algorithm B*, are shown in Figures 2.13 and 2.14.

```
procedure PLAN($\Pi, n$)                          ▷ Planning problem $\Pi$ on $n$ processes
    $P := \{\Pi_1, \ldots, \Pi_n\}$;               ▷ encodings of $\Pi$ in propositional logic
    $M := \emptyset$;
    repeat
        $P' := P$;
        for all $\Pi_i \in P'$ do
            continue execution of $\Pi$ for $\epsilon$ seconds;
            if evaluation of $\Pi$ is terminated then
                $n := n + 1$;
                $P := P \bigcup \{\Pi_n\} \backslash \{\Pi_i\}$;
                if $\Pi_i$ is satisfied then $M :=$ satisfying model for $\Pi_i$; end if
            end if
        end for
    until $M \neq \emptyset$
    return $M$;
end procedure
```

Figure 2.13: Algorthim A for Planning as Satisfiability, using n parallel processes.

The idea behind both algorithms is that several encodings can be run in parallel. Once an encoding is found to be satisfiable, meaning that a plan has been found, all processes can be terminated. This will avoid spending time on the hardest unsatisfiable instances – as in the sequential approach – at the cost of optimality guarantees. Rintanen notes that in some cases the earliest satisfiable encodings are not cheaper than the unsatisfiable ones. Rintanen argues that in many settings optimality is not required, or that a different kind of optimality, such as number of actions, is required and in a parallel plan makespan optimality does not imply a good quality plan.

Rintanen [82] shows that Algorithm A can lead to significant speed ups as the number of parallel processes increases. Algorithm B is much more stable, finding solutions faster that Algorithm A for low values of n and γ. The value of n in Algorithm A has a large impact on the solution time. In contrast the choice of γ in Algorithm B has much smaller significance, avoiding the difficulty in choosing n.

Streeter and Smith [95] introduced a method for using decision procedures efficiently for optimization, experimenting upon both Planning and job shop scheduling. The approach makes use of the notion that the more difficult instances are the closest to the optimal bound and the time to solve for other instances falls off evenly as the bound increases or decreases. Similar to Algorithm A presented by Rintanen [82], the query to the SAT solver is only given a certain amount of time. Streeter and Smith [95] gradually increase this time limit in conjunction with smart upper and lower bounds to home in on an optimal solution. The algorithm is shown in Figure 2.15.

Streeter and Smith [95] generalised this algorithm to allow for varied time increments; choice of k; and balance between exploring the upper and lower bounds.

```
procedure PLAN($\Pi, \gamma$)                              ▷ Planning problem $\Pi$
    $P := \{\Pi_1, \ldots, \Pi_n\}$;             ▷ encodings of $\Pi$ in propositional logic
    $T := \{t_1 \ldots, t_n\}$, $t_i := 0$, $\forall t_i \in T$;
    $M := \emptyset$;
    $t := 0$;
    repeat
        $P' := P$;
        $T' := T$;
        $t := t + \delta$;
        for all $\Pi_i \in P'$ do
            if $t_i + n\epsilon \leq t\gamma^i$ for some maximal $n \geq 1$ then
                continue execution of $\Pi$ for $n\epsilon$ seconds;
                $t_i := t_i + n\epsilon$;
                if evaluation of $\Pi$ is terminated then
                    $n := n + 1$;
                    $P := P \bigcup \{\Pi_n\} \backslash \{\Pi_i\}$;
                    $T' := T \bigcup \{t_n\} \backslash \{t_i\}$, $t_n := 0$;
                    if $\Pi_i$ is satisfied then
                        $M :=$ satisfying model for $\Pi_i$;
                    end if
                end if
            end if
        end for
    until $M \neq \emptyset$
    return $M$;
end procedure
```

Figure 2.14: Algorthim B for Planning as Satisfiability, using geometric division of CPU use based on parameter γ where δ is some time increment.

```
1: procedure PLAN($\Pi, U$)                ▷ Planning problem $\Pi$ with upper bound $U$
2:     $T := 2$, $l := 1$, $u := U$;
3:     $t_l := \infty$, $t_u := -\infty$;
4:     while $l < u$ do
5:         if $[l, u-1] \subseteq [t_l, t_u]$ then
6:             $T := 2T$;
7:             $t_l := \infty$, $t_u := -\infty$;
8:         end if
9:         $u' = u - 1$;
10:        $k := \begin{cases} \frac{l+u'}{2} & \text{if } [l,u'] \text{and } [t_l,t_u] \text{are disjoint or } t_l = \infty \\ \frac{l+t_l-1}{2} & \text{if } [l,u'] \text{and } [t_l,t_u] \text{intersect and } t_l - l > u' - t_u \\ \frac{t_u+1+u'}{2} & \text{if } [l,u'] \text{otherwise;} \end{cases}$
11:        execute $\Pi_k$ for $T$ seconds;        ▷ encoding of $\Pi$ in propositional logic
12:        if $\Pi_i$ is satisfied then
13:            $M :=$ satisfying model for $\Pi_k$;
14:            $u := k$;
15:        else if $\Pi$ is unsatisfiable then
16:            $l := k + 1$;
17:        else
18:            $t_l := min(t_l, k)$, $t_u := max(k, t_u)$;
19:        end if
20:    end while
21:    return $M$;
22: end procedure
```

Figure 2.15: A simple top-level strategy for optimisation proposed by Streeter and Smith [95].

2.2 Model-Checking and Planning with Quantified Boolean Formulae

This section reviews the current use of QBF with respect Planning and reachability. This is largely limited to two approaches: Planning under uncertainty and Planning as Model-Checking. One exception is provided by Rintanen [80]. Rintanen described a QBF encoding of reachability that is logarithmic in the number of time-steps. This encoding was briefy described in Section 1.3 and is formalised in Chapter 3.

2.2.1 Planning Under Uncertainty

Until now QBF has been considered in the context of Conformant Planning [79], but not as a practical approach to classical Planning. Conformant Planning is the problem of finding conditional plans in a setting where the initial state is unknown, only partially known, or action effects are non-deterministic [43]. We can assume without any loss of expressivity that all action effects are deterministic and all uncertainty is contained in the initial state.

With such an assumption, a Conformant Planning problem is the 4-tuple $\langle \mathcal{F}, \mathcal{A}, I, G \rangle$ in which:

- $\mathcal{F}$ is a set of observable *fluents* that decide how plan execution proceeds;
- $\mathcal{A}$ is a set of *actions*;
- I is a formula over $\mathcal{F}$, describing the set of initial states; and
- G is a formula over $\mathcal{F}$ representing the goal.

A conditional plan determines for all possible initial states an execution that reaches a state S, such that $S \models G$.

Rintanen [79] showed that Conformant Planning is Π_2^p-hard by providing a translation from any QBF with quantifier prefix $\forall\exists$ to Conformant Planning. Rintanen outlined several translations from Conformant Planning to QBF with a $\exists\forall\exists$ prefix, later optimising the encoding to drop the outermost existential quantifier [83].

The encodings use the quantifier prefix:

$$\exists P \forall C \exists E$$

where: P contains variables that represent the plan; C the universal variables that enumerate the uncertainty in the initial state; and E the variables responsible for evaluating the execution of a plan. Rintanen presented a number of encodings using this quantifier prefix, all of which use the idea of embedding within the translation a finite-state-automata that dictates which actions are to be applied, according to conditions upon the observable fluents of the domain. In each case the universal quantification afforded by the QBF is used to represent the non-determinism – either by using auxiliary variables that map to balanced pairs of conditions in the FSM, or by directly representing the fluents of the initial state.

Giunchiglia [39] proposed a SAT-based procedure for Conformant Planning, which was further simplified and extended by Ferraris and Giunchiglia [28]. Similar to the approaches presented in this work, Ferraris and Giunchilglia made it clear that their approach was applicable with any action representation and Planning strategy. Although the translations were to SAT, a universal quantification is implicit in the top-level strategy, and is made explicit in later work by Rintanen [83] as a translation to QBF.

De Luca et al. [21] used a QBF representation in order to capture safety constraints in a possibly non-deterministic setting. The safety constraints were described by Quantified Linear Temporal Logic (QLTL) [30]. The encoding used the extra expressivity of the QBF to represent these safety properties. During the Planning process, all possible plans are enumerated – corresponding to all possible satisfying assignments to the QBF – then checked for validity and safety using the techniques described by Giunchiglia [39].

These techniques for incorporating non-determinism into QBF encodings of Planning problems can be applied to the translations described in this work. However, this area of research is otherwise orthogonal to ours, and we present encodings only for classical, deterministic Planning.

2.2.2 Model-Checking with QBF

Model-Checking [9, 18] is concerned with the automatic verification of programs and/or specifications. Similar to Planning, Model Checking is an automatic state exploration and as such there exists the possibility for cross-fertilization between these two areas of research; Giunchiglia and Traverso [42] and Magazzeni [62] have implemented Planning systems based on Model Checking. In fact Biere et. al. [6] have shown that it is possible to apply the DPLL algorithm to Model Checking problems, building on the work of Kautz et. al. [55, 56] described earlier.

Biere et al. [7] noted that Binary Decision Diagrams (BDDs), traditionally used as the underlying expression for symbolic model checkers [69], suffer from space efficiency issues. Motivated by this, Biere et al. introduced semantics for Bounded Model-Checking (BMC), translating the Model-Checking problem to a SAT instance that is linear in the size of the domain, and quadratic in the time-step bound. These translations resemble the state-based encodings of Kautz et al. [55, 56], with the addition of Linear Temporal Logic formulae and loops. They note that the translation is a first step in applying SAT procedures to symbolic Model-Checking, in particular:

> *...it would be interesting to study efficient decision procedures for QBF. Combining bounded model checking with other state space reduction techniques presents another interesting problem.*

which is the subject of this thesis.

Dershowitz et al. evaluated the use of QBF in BMC [22], using an encoding logarithmic in the number of time-steps – similar to the encodings presented in Chapter 3. The encoding they used is described by Papadimitriou as a reduction of Reachability to QBF [74], the basic idea being to encode only a single copy of the transition relation, using universal quantification to apply it to all states:

$$\begin{aligned}
&R_k(X_0, X_k) := \exists X_1, \dots, X_{k-1} (\\
&\quad I(X_0) \wedge G(X_k) \wedge \forall U, V (\\
&\qquad (\textstyle\bigvee_{i=0}^{k-1} (U \leftrightarrow X_i) \wedge (V \leftrightarrow Z_{i+1})) \rightarrow \tau(U, V)\\
&\qquad))
\end{aligned}$$

The disadvantage of this encoding is that it contains more variables than an equivalent SAT encoding, and that there are a large number of equality constraints, offsetting the advantage gained from reducing the number of transitions. General-purpose QBF solvers find these encodings extremely difficult, for instance, QuBE [41] only solves a small number of the generated problems. Dershowitz et al. implemented a specialized QBF solver, jSAT, that was able to solve the BMC problems much faster, and although not as fast as comparable SAT translations, avoiding the space overhead of SAT.

In comparison to this approach, the QBF encodings logarithmic in the number of time-steps presented in Chapter 3 differ greatly in construction, reducing the number of variables as well as clauses. In addition, by avoiding universal quantification of states, they are easier to solve. We apply general-purpose QBF solvers to these encodings, achieving similar results to Dershowitz et al.

2.3 Solving Quantified Boolean Formulae

Research into solving QBF lacks the maturity of the research into solving SAT; however there are a large number of solvers boasting different approaches to the problem. This section will outline the most common methods, which are used in the experiments in Chapter 5. The solvers can be found in the QBFLIB [38] (QBFLIB is a collection of instances, solvers, and tools related to Quantified Boolean Formulae). Knowledge of these algorithms will give insight into the performance of the various QBF encodings of Planning problems on these solvers.

2.3.1 DPLL-based Solvers

Cadoli et al. introduced a *top-down* technique for solving QBFs closely related to the DPLL procedure for SAT [10]. The algorithm, now commonly known as QDPLL, is guaranteed to work in polynomial space. The algorithm closely resembles that of DPLL, described in Figure 2.8. The main differences are the choice of branch variable; propagation techniques, which will not be discussed here; and treatment of sub-formulae upon a branch.

Branching on a variable, as described by line 9 in Figure 2.8, is performed differently depending upon the quantification of that variable. Given a set of clauses Π and variable v such that $v \in \Pi \vee \neg v \in \Pi$. If the variable v is existentially quantified the procedure (QDPLL) recurses as:

$$QDPLL(\Pi \cup \{v\}) \vee QDP(\Pi \cup \{\neg v\});$$

whereas if v is quantified universally the procedure recurses as:

$$QDPLL(\Pi \cup \{v\}) \wedge QDP(\Pi \cup \{\neg v\}).$$

The choice of branch variable in QDPLL is restricted to the outermost quantified variable, or a variable belonging to the outermost quantified set. QDPLL solvers are known as top-down solvers due to this behaviour. This is the most significant difference when considering how the algorithm performs upon the encodings presented in this work. In a SAT encoding of a Planning problem there is no restriction on the choice of branch variable; the search can be directed at any part of the plan and at any time-step. With the QBF encodings presented

problem	1	2	3	4	5	6	7	8	9	10
variables	302	437	482	581	677	393	983	1055	1436	1817
resolved	25	0	0	0	0	65	0	0	0	18

Table 2.1: Variables resolved by sQueezeBF using Q-resolution per problem in the *driverlog* domain.

here the search is restricted in scope. This restriction affects the encodings presented in Chapter 3 and Chapter 4 differently.

In the encodings presented in Chapter 4 the restriction has little impact. The search is restricted to a portion of each state, but not restricted in terms of time-steps. Propagation is unhindered inside each state.

The encodings presented in Chapter 3, those logarithmic in the number of time-steps, are more seriously affected. Due to the structure of these encodings, the top-down algorithm is restricted to only branching upon variables that represent the mid-point state of the plan. Once an assignment is made to every variable in the outermost set the algorithm recurses on the first and second half of the plan. This means that the information in the initial state and goal state is not propagated until the innermost existential states are reached. Intuitively the algorithm can be described as guessing each even-numbered state and backtracking in the event that they do not immediately form a plan. For this reason the top-down solvers perform abnormally poorly on the encodings presented in Chapter 3, scaling poorly with the number of quantifier alternations and exhibiting a large variance in memory use and time to solve. The structure of these encodings is described in Chapter 3; the performance of two top-down solvers is examined in Chapter 5.

We use two QDPLL-based search solvers: QuBE7.0 [41] and DepQBF (v0.1) [61] both with conflict-driven clause learning – as in DPLL – and solution-driven cube learning. QuBE6.1, the predecessor to QuBE7.0 developed by Giunchiglia et al. was the strongest mono-engine solver in the 2008 QBF Evaluation [38]. Before QuBE7.0 begins the search process it runs a preprocessing step with the preprocessor sQueezeBF [40]. This preprocessor applies four different operations to the input formula:

1. propagating unit and pure literals, and eliminating subsumed clauses;
2. identifying variables that are defined by a logical combination of other variables;
3. processing equivalences; and
4. performing Q-resolution.

Operations 1, 2, and 3 have no effect on the encodings presented in this work. Operation 4, Q-resolution, has a small impact, illustrated by Table 2.1. This table shows the number of variables simplified by Q-resolution on problems in the *driverlog* domain.

Q-resolution is performed by removing an existential variable x from the problem by replacing the set of clauses where x appears positively, S_x, and the set of clauses where x appears negatively, $S_{\neg x}$ with the set $S_x \cdot S_{\neg x}$ obtained by resolving on x all the pairs of clauses from S_x and $S_{\neg x}$. For example, consider the clauses:

$$S_v : \{(\neg r \vee v),\ (s \vee v),\ (x \vee y \vee v)\}$$
$$S_{\neg v} : \{(\neg v \vee r),\ (\neg v \vee \neg x \vee \neg y \vee r)\}.$$

Resolving on all six resolutions between pairs produces three trivial resolvents, the others are:

$$(s \vee r)$$
$$(x \vee y \vee r)$$
$$(s \vee \neg x \vee \neg y \vee r).$$

Q-resolution was introduced by QUANTOR [4].

DEPQBF (v0.1) [60] participated in the 2010 QBF Evaluation [38] and was placed first according to a score-based ranking. The key feature of DEPQBF is the inclusion of dependency schemes as compact dependency graphs. A number of techniques, not specific to the solver, are also included, such as: watched data structures; removal of learned constraints; and restarts.

2.3.2 Non-QDPLL Solvers

We use two non-QDPLL solvers: QUANTOR (v3.0) [4], a resolution-based solver; and CIRQIT2.1 [44], a circuit-based QBF solver.

When solving a QBF in prenex cnf, the quantifier order of variables must be respected. The QDPLL approach works from the outermost to innermost variables in a top-down manner. QUANTOR, implemented by Biere [4], instead uses a *bottom-up* approach, working from the innermost to outermost quantified variables.

Existential variables are removed using Q-resolution and universal variables are removed through expansion. Q-resolution alone, although complete, is impractical due to the large number of clauses it generates. In QUANTOR memory use is carefully monitored and only the cheapest variables are picked for resolution, and only from the innermost existentially quantified set. In addition, Q-resolution is only invoked when the size of the formula will not increase too much, otherwise universal variable are expanded, as described in Section 1.2.

CIRQIT2.1 [44] solved the most problems in the non-prenex non-cnf track of the 2010 QBF Evaluation [38] and applies QBF solving techniques, such as clause and cube learning, to a circuit based representation. Specifically CIRQIT operates upon the ISCAS-85 format along with a quantifier prefix, solving the circuit using DPLL search. An encoding of Planning problems into ISCAS-85 was implemented, equivalent to the QBF encodings logarithmic in time-steps presented in Chapter 3. The ISCAS-85 formalism has the potential to be a more compact representation for QBF problems. However, the solver does not perform well on the encodings of Planning problems, as is explained in more detail in Chapter 5.

Chapter 3

QBF Encodings with Exponential Time-steps

In this Chapter we will describe two translations from Planning to QBF. Both translations share the property that the number of variable sets representing a state is logarithmic in the number of time-steps represented by the formula. In other words, some variables are reused to represent multiple states in the plan.

Figures 3.1, 3.2 and 3.3 illustrate the intuition behind these encodings. In the figures the square boxes represent sets of existentially quantified variables, the circles represent a universally quantified variable. The nodes representing universally quantified variables have two branches, corresponding to the two subformulae produced by making an assignment. The aggregation of all existentially quantified sets in each diagram corresponds to the expansion of all its universal variables, as described in Section 1.2.

Figure 3.1 shows the expansion of the Flat Encoding, described in Section 3.1. Each existentially quantified set X represents a state in the planning problem, in exactly the same way as in formulae 1.1, reproduced here for convenience:

$$I(X_1) \wedge \bigwedge_{i=1}^{n+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge G(X_{n+1}) \qquad (n \geq 0) \qquad (1.1)$$

X_{st} represents the midpoint of the plan. The two subformulae obtained through the expansion of y represent the first and second half of the plan. Figure 3.2 shows the constraints between these existential sets that enforce equality between states, or the transition constraints. The equality constraint $X_\alpha \leftrightarrow X_\beta$ ensures that: $x_\alpha \leftrightarrow x_\beta$ for each $x_\alpha \in X_\alpha$ with corresponding $x_\beta \in X_\beta$.

As can be seen from this figure, two transitions are made, one at each leaf node of the tree. As each leaf node is actually the same existential variables copied due to expansion it is necessary to only state the transition constraints once. By recursing with this formulation a number of transitions can be described equal to 2^k where k is the depth of the tree.

The intuition behind the second encoding, the Compact Tree Encoding described in Section 3.2, is to use the same recursive formulation, but remove the redundancy. This means that each combination of existentially quantified

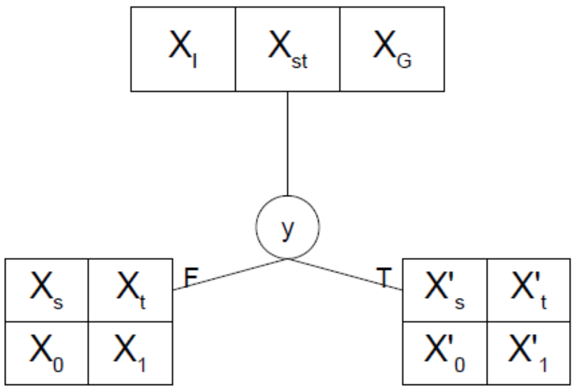


Figure 3.1: The tree formed by expanding universal quantifiers in the Flat Encoding ($k = 1$) with a generic state-based representation. Each box corresponds to a set of existentially quantified variables; each circle a universally quantified variable. Variables beneath the universal variable are copied to represent its expansion.

variable set and context prescribed by assignments to universally quantified variables will correspond to a unique state in the final plan. Figure 3.3 shows the expansion of the Compact Tree Encoding. The basic structure of this encodings resembles that of the Flat Encoding: the outermost existentially quantified variable, X_0, represents the midpoint of the plan and the two subformulae obtained through the expansion of y represent the first and second half of the plan. However, there are some important differences:

- the transitions exist between each node in the tree, rather than at each leaf node; and
- the encoding uses fewer variable sets to represent a larger number of states.

The encodings are described in more detail below, then compared in Chapter 5.

3.1 Flat Encoding

Rintanen [80] introduced Flat Encoding as an approach to general reachability. It is presented here specific to Planning. Consider the formula

$$I(X_I) \land E_k(X_I, X_G) \land G(X_G) \tag{3.1}$$

where $k \geq 0$ and represents the folding parameter. In the following, given two finite sets X_α and X_β of variables, $\exists X_\alpha X_\beta$ denotes the result of existentially quantifying each variable in $X_\alpha \cup X_\beta$, and

$$X_\alpha \leftrightarrow X_\beta$$

stands for

$$\bigwedge_{x \in X} (x_\alpha \leftrightarrow x_\beta).$$

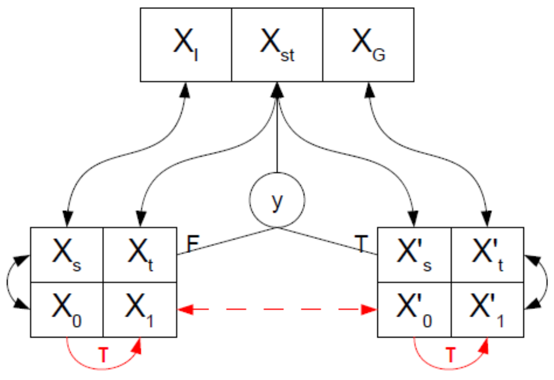


Figure 3.2: The tree formed by expanding universal quantifiers in the Flat Encoding ($k = 1$) with a generic state-based representation. Equality constraints are represented by dark arrows, transition constraints by red arrows with the label τ. The dashed arrow shows a pair of states that are made implicitly equal.

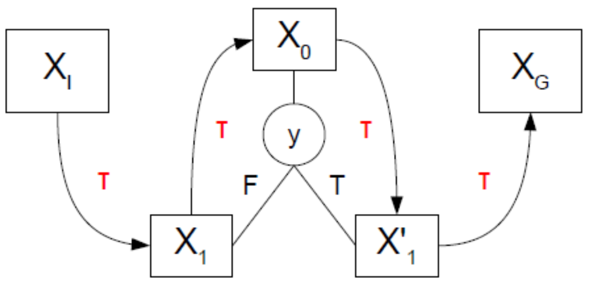


Figure 3.3: The tree formed by expanding universal quantifiers in the Compact Tree Encoding with a generic state-based representation. Each box corresponds to a set of existentially quantified variables; each circle a universally quantified variable. Variables beneath the universal variable are copied to represent its expansion. Transition constraints are shown by arrows with the label τ.

$E_k(X_I, X_G)$ is defined as follows:

$$\begin{aligned}
E_k(X_I, X_G) := {} & \exists X_{st} \forall y \exists X_s X_t (\\
& (\neg y \Rightarrow ((X_{st} \leftrightarrow X_t) \wedge (X_I \leftrightarrow X_s))) \\
& \wedge \\
& (y \Rightarrow ((X_{st} \leftrightarrow X_s) \wedge (X_G \leftrightarrow X_t))) \\
& \wedge \\
& E_{k-1}(X_s, X_t) \qquad)
\end{aligned}$$

if $k > 0$, and

$$\begin{aligned}
& E_0(X_I, X_G) := \exists X_1 \exists X_2 \\
& ((X_I \leftrightarrow X_1) \wedge \sigma(X_1) \wedge \tau(X_1, X_2) \wedge \sigma(X_2) \wedge (X_2 \leftrightarrow X_G))
\end{aligned}$$

when $k = 0$.

The correspondence between the basic SAT formula (1.1) and the Flat Encoding is clear when $k = 0$. When $k > 0$, by expanding the universal quantifiers, we find again a correspondence between the two formulae, as established by the following proposition and proof.

Proposition 1 *For each $k \geq 0$, if $n = 2^k$ the existential closures of (1.1) and (3.1) are equivalent.*

Proof: We prove, by induction on k, that $E_k(X_I, X_G)$ is equivalent to

$$\begin{array}{l}\exists X_1 \ldots \exists X_{n+1} \\ \quad ((X_I \leftrightarrow X_1) \wedge \bigwedge_{i=1}^{n+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge (X_{n+1} \leftrightarrow X_G))\end{array}$$

with $n = 2^k$.

<u>$k = 0$</u>**:** In this case $n = 1$ and the proposition trivially holds.

<u>$k = m + 1$</u>**:** By expanding the outermost universal quantifier, $E_k(X_I, X_G)$ becomes

$$\begin{array}{l}E_{m+1}(X_I, X_G) = \exists X_{st}(\\ \exists X_s X_t(\\ ((X_I \leftrightarrow X_s) \wedge (X_{st} \leftrightarrow X_t))\wedge \\ E_m(X_s, X_t)) \\ \wedge \\ \exists X'_s X'_t(\\ ((X_{st} \leftrightarrow X'_s) \wedge (X_G \leftrightarrow X'_t))\wedge \\ E_m(X'_s, X'_t)) \quad)\end{array}$$

By the induction hypothesis this is equivalent to:

$$\begin{array}{l}\exists X_{st}(\\ \exists X_1 \ldots \exists X_{2^m+1}(\\ \quad (X_I \leftrightarrow X_1) \wedge \bigwedge_{i=1}^{2^m+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{2^m} \tau(X_i, X_{i+1}) \wedge (X_{2^m+1} \leftrightarrow X_{st})) \\ \wedge \\ \exists X'_1 \ldots \exists X'_{2^m+1}(\\ \quad (X_{st} \leftrightarrow X'_1) \wedge \bigwedge_{i=1}^{2^m+1} \sigma(X'_i) \wedge \bigwedge_{i=1}^{2^m} \tau(X'_i, X'_{i+1}) \wedge (X'_{2^m+1} \leftrightarrow X_G)) \quad)\end{array}$$

which can be rewritten as

$$\begin{array}{l}\exists X_1 \ldots \exists X_{2^{m+1}+1}(\\ \quad (X_I \leftrightarrow X_1) \wedge \bigwedge_{i=1}^{2^{m+1}+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{2^{m+1}} \tau(X_i, X_{i+1}) \wedge (X_{n+1} \leftrightarrow X_G))\end{array}$$

which is the proposition. $\square$

The above formulation involves $(3k+2)|X|$ existential variables and k universal variables. Further, the Flat Encoding can be converted into prenex conjunctive normal form (corresponding to the QDIMACS format used by most QBF solvers) with $4(2k+1)|X| + |\tau| + 2|\sigma|$ clauses, where $|\tau|$ is the number of clauses in the transition relation and $|\sigma|$ the number of clauses in the state constraints of the original Planning problem.

3.2 Compact Tree Encoding

This section describes a new encoding that removes redundancy and requires considerably fewer variables than the Flat Encoding.

Given a QBF in the Compact Tree Encoding with k universal quantifiers, the expansion corresponding to the QBF forms a tree of depth k and removes all redundancy from the formula by only specifying equivalent states once. The key novelty of this encoding lies in the traversal of its tree structure, in which transitions are encoded from each leaf node to one of the nodes in the preceding layers of the tree. This leads to a formula that encodes $2^{k+1} - 2$ transitions in a tree with k layers. The transition constraints are only applicable under certain contexts, described by assignments to universal variables. The formula is quadratic in k because every transition to and from level i requires $k - i - 1$ terms to express the context. The Flat Encoding does not require these contexts and therefore is linear in k. However, twice as many variables are required to enforce equivalence of sets of existentially quantified variables on different branches as are required in our second encoding to describe the traversal of the tree.

Intuitively, the Flat Encoding describes a one-to-one correspondence between the states traversed and the *leaves* of the expansion corresponding to the QBF. By contrast, in our second encoding, which we call a *tree-structured* encoding, there is a one-to-one correspondence between the states traversed and the *nodes* of the tree corresponding to the QBF. Every existentially quantified layer in the QBF is used to represent at least one distinct state.

The encoding in question can be written as the formula:

$$I(X_I) \wedge Q_k(X_I, X_G) \wedge G(X_G) \tag{3.2}$$

where

$$\begin{aligned}
&Q_k(X_I, X_G) := \exists X_k \forall y_k \ldots \exists X_1 \forall y_1 \exists X(\\
&\quad (\sigma(X) \wedge \textstyle\bigwedge_{i=1}^{k} \sigma(X_i)) \\
&\quad \wedge \\
&\quad ((\textstyle\bigwedge_{i=1}^{k} \neg y_i) \Rightarrow (X_I \leftrightarrow X)) \\
&\quad \wedge \\
&\quad ((\textstyle\bigwedge_{i=1}^{k} y_i) \Rightarrow (X \leftrightarrow X_G)) \\
&\quad \wedge \\
&\quad \textstyle\bigwedge_{i=1}^{k}(\quad ((\neg y_i \wedge \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i)) \\
&\quad \qquad\qquad \wedge \\
&\quad \qquad\qquad ((y_i \wedge \textstyle\bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X)) \quad))
\end{aligned}$$

This formula states that the goal state X_G is reachable in $2^{k+1} - 2$ applications of the transition relation. Only $k + 1$ states are quantified (X_k to X_1 and X).

The Compact Tree Encoding (3.2) has k universal variables and $(k + 1)|X|$ existential variables. Further, with (3.2) we check the existence of plans having makespan equal to $2^{k+1} - 1$, i.e. twice the makespan allowed by the Flat Encoding. However, the conversion of (3.2) to prenex conjunctive normal form has $2k|\tau| + (k + 1)|\sigma|$ clauses.

Proposition 2 *For each $k \geq 0$, if $n = 2^{k+1}$ the existential closures of (1.1) and (3.2) are equivalent.*

Proof: We prove, by induction on k, that $Q_k(X_I, X_G)$ is equivalent to

$$\begin{aligned}&\exists X_1 \ldots \exists X_{n+1}\\ &\quad ((X_I \leftrightarrow X_1) \wedge \textstyle\bigwedge_{i=1}^{n+1} \sigma(X_i) \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge (X_{n+1} \leftrightarrow X_G))\end{aligned}$$

with $n = 2^{k+1} - 2$.

<u>$k = 0$</u>**:** In this case, $Q_k(X_I, X_G)$ becomes

$$\exists X((X_I \leftrightarrow X) \wedge \sigma(X) \wedge (X \leftrightarrow X_G))$$

and the proposition trivially holds.

<u>$k = p + 1$</u>**:** Expanding the outermost universal variable, y_{p+1}:

The goal and initial state constraints

$$\begin{array}{l}((\bigwedge_{i=1}^{p+1} \neg y_i) \Rightarrow (X_I \leftrightarrow X))\\ \wedge\\ ((\bigwedge_{i=1}^{p+1} y_i) \Rightarrow (X \leftrightarrow X_G))\end{array}$$

each appear in only one half of the expansion. Additionally the transition constraints

$$\bigwedge_{i=1}^{p+1}\left(\begin{array}{l}((\neg y_i \wedge \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i))\\ \wedge\\ ((y_i \wedge \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X))\end{array}\right)$$

when $i = p + 1$ both produce one constraint that is only applicable in one branch of the expansion:

$$(\bigwedge_{i=1}^{p} \neg y_i) \Rightarrow \tau(X_{p+1}, X)$$

when $y_i \models \top$ and

$$(\bigwedge_{i=1}^{p} y_i) \Rightarrow \tau(X, X_{p+1})$$

when $y_i \models \bot$. The remaining constraints when $i < p + 1$ become

$$\bigwedge_{i=1}^{p}\left(\begin{array}{l}((\neg y_i \wedge \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i))\\ \wedge\\ ((y_i \wedge \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X))\end{array}\right).$$

Therefore, after expansion $Q_k(X_I, X_G)$ becomes

$$\begin{array}{l}
\exists X_{p+1}(\\
\quad \exists X_p \forall y_p \ldots \exists X_1 \forall y_1 \exists X(\\
\qquad (\sigma(X) \wedge \bigwedge_{i=1}^{p} \sigma(X_i)) \\
\qquad \wedge((\bigwedge_{i=1}^{p} \neg y_i) \Rightarrow (X_I \leftrightarrow X)) \\
\qquad \wedge((\bigwedge_{i=1}^{p} y_i) \Rightarrow \tau(X, X_{p+1})) \\
\qquad \wedge \bigwedge_{i=1}^{p}(\ ((\neg y_i \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i)) \\
\qquad\qquad\qquad \wedge((y_i \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X)) \quad)) \\
\quad \exists X'_p \forall y'_p \ldots \exists X'_1 \forall y'_1 \exists X'(\\
\qquad (\sigma(X') \wedge \bigwedge_{i=1}^{p} \sigma(X'_i)) \\
\qquad \wedge((\bigwedge_{i=1}^{p} \neg y'_i) \Rightarrow \tau(X_{p+1}, X')) \\
\qquad \wedge((\bigwedge_{i=1}^{p} y'_i) \Rightarrow (X' \leftrightarrow X_G)) \\
\qquad \wedge \bigwedge_{i=1}^{p}(\ ((\neg y'_i \bigwedge_{j=1}^{i-1} y'_j) \Rightarrow \tau(X', X'_i)) \\
\qquad\qquad\qquad \wedge((y'_i \bigwedge_{j=1}^{i-1} \neg y'_j) \Rightarrow \tau(X'_i, X')) \quad))).
\end{array}$$

The first half of the expansion,

$$\begin{array}{l}
\exists X_{p+1}(\\
\quad \exists X_p \forall y_p \ldots \exists X_1 \forall y_1 \exists X(\\
\qquad (\sigma(X) \wedge \bigwedge_{i=1}^{p} \sigma(X_i)) \\
\qquad \wedge((\bigwedge_{i=1}^{p} \neg y_i) \Rightarrow (X_I \leftrightarrow X)) \\
\qquad \wedge((\bigwedge_{i=1}^{p} y_i) \Rightarrow \tau(X, X_{p+1})) \\
\qquad \wedge \bigwedge_{i=1}^{p}(\ ((\neg y_i \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i)) \\
\qquad\qquad\qquad \wedge((y_i \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X)) \quad)))
\end{array}$$

can be rewritten to include an equivalence relation, and so conforms to the form presented in (3.2).

$$\begin{array}{l}
\exists X_{p+1}, X'_{p+1}(\ \tau(X'_{p+1}, X_{p+1}) \wedge \\
\qquad\qquad \exists X_p \forall y_p \ldots \exists X_1 \forall y_1 \exists X(\\
\qquad\qquad\quad (\sigma(X) \wedge \bigwedge_{i=1}^{p} \sigma(X_i)) \\
\qquad\qquad\quad \wedge((\bigwedge_{i=1}^{p} \neg y_i) \Rightarrow (X_I \leftrightarrow X)) \\
\qquad\qquad\quad \wedge((\bigwedge_{i=1}^{p} y_i) \Rightarrow (X \leftrightarrow X'_{p+1})) \\
\qquad\qquad\quad \wedge \bigwedge_{i=1}^{p}(\ ((\neg y_i \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau(X, X_i)) \\
\qquad\qquad\qquad\qquad \wedge((y_i \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau(X_i, X)) \quad)))
\end{array}$$

and so, by induction hypothesis, is equivalent to

$$\begin{array}{l}
\exists X_{p+1}, X'_{p+1}(\\
\quad \tau(X'_{p+1}, X_{p+1}) \wedge \\
\quad \exists X_1 \ldots \exists X_{2^{p+1}-1}(\\
\qquad (X_I \leftrightarrow X_1) \wedge \bigwedge_{i=1}^{2^{p+1}-1} \sigma(X_i) \\
\qquad \wedge \bigwedge_{i=1}^{2^{p+1}-2} \tau(X_i, X_{i+1}) \wedge (X_{2^{p+1}-1} \leftrightarrow X'_{p+1})).
\end{array}$$

The second half of the expansion can be equated to a similar expression in an analogous fashion – with the exception that the chain of transitions are between X_{p+1} and X_G, as they represent the second half of the plan.

Thus, we have

$$
\begin{aligned}
&\exists X_{p+1}(\\
&\quad \exists X'_{p+1}, X_1 \ldots \exists X_{2^{p+1}-1}(\\
&\quad\quad \tau(X'_{p+1}, X_{p+1})\\
&\quad\quad \wedge (X_I \leftrightarrow X_1) \wedge \textstyle\bigwedge_{i=1}^{2^{p+1}-1} \sigma(X_i)\\
&\quad\quad \wedge \textstyle\bigwedge_{i=1}^{2^{p+1}-2} \tau(X_i, X_{i+1}) \wedge (X_{2^{p+1}-1} \leftrightarrow X'_{p+1}))\\
&\quad \exists X''_{p+1}, X'_1 \ldots \exists X'_{2^{p+1}-1}(\\
&\quad\quad \tau(X_{p+1}, X''_{p+1})\\
&\quad\quad \wedge (X''_{p+1} \leftrightarrow X'_1) \wedge \textstyle\bigwedge_{i=1}^{2^{p+1}-1} \sigma(X'_i)\\
&\quad\quad \wedge \textstyle\bigwedge_{i=1}^{2^{p+1}-2} \tau(X'_i, X'_{i+1}) \wedge (X'_{2^{p+1}-1} \leftrightarrow X_G))\\
&)
\end{aligned}
$$

and by combining these transition chains we arrive at

$$
\begin{aligned}
&\exists X_1 \ldots \exists X_{2^{p+2}-1}(\\
&\quad (X_I \leftrightarrow X_1) \wedge \textstyle\bigwedge_{i=1}^{2^{p+2}-1} \sigma(X_i) \wedge \textstyle\bigwedge_{i=1}^{2^{p+2}-2} \tau(X_i, X_{i+1}) \wedge (X_{2^{p+2}-1} \leftrightarrow X_G))
\end{aligned}
$$

which is the proposition. □

3.3 Comparison between Flat and Compact Tree Encoding

The following simple abstract example emphasises the difference between the Compact Tree Encoding and the Flat Encoding.

If we build an expression containing two universal quantifiers using the Compact Tree Encoding, we express the existence of a plan of makespan 8. Below is a simple example of this, excluding the state constraints for readability.

$$
\begin{aligned}
&\exists X_2 \forall y_2 \exists X_1 \forall y_1 \exists X(\\
&\quad (\neg y_2 \wedge \neg y_1 \Rightarrow \tau(I, X))\\
&\quad \wedge (y_2 \wedge y_1 \Rightarrow \tau(X, G))\\
&\quad \wedge (\neg y_2 \wedge y_1 \Rightarrow \tau(X, X_2))\\
&\quad \wedge (y_2 \wedge \neg y_1 \Rightarrow \tau(X_2, X))\\
&\quad \wedge (\neg y_1 \Rightarrow \tau(X, X_1))\\
&\quad \wedge (y_1 \Rightarrow \tau(X_1, X))).
\end{aligned}
$$

It should be noted that the last two transitions are both invoked twice: once in the context where y_1 is true and once in the context where it is false. The other transitions are all invoked once, accounting for all 8 transitions.

By contrast, two universally quantified variables using the Flat Encoding produces:

$$
\begin{aligned}
&\exists X_1 \forall y_1 \exists X_2 \exists X_3(\\
&\quad (y_1 \Rightarrow (I \leftrightarrow X_2) \wedge (X_1 \leftrightarrow X_3))\\
&\quad \wedge (\neg y_1 \Rightarrow (X_1 \leftrightarrow X_2) \wedge (X_3 \leftrightarrow G))\\
&\quad\ \exists X_4 \forall y_2 \exists X_5 \exists X_6(\\
&\quad\quad (y_2 \Rightarrow (X_2 \leftrightarrow X_5) \wedge (X_4 \leftrightarrow X_6))\\
&\quad\quad \wedge (\neg y_2 \Rightarrow (X_4 \leftrightarrow X_5) \wedge (X_3 \leftrightarrow X_6))\\
&\quad\quad\ \exists X_7 \exists x_8(\\
&\quad\quad\quad (X_5 \leftrightarrow X_7) \wedge (X_8 \leftrightarrow X_6) \wedge \tau(X_7, X_8)))).
\end{aligned}
$$

The single transition is invoked in all of the contexts generated by assignments to y_1 and y_2. This is only 4 contexts, so the encoding expresses the existence of a plan of makespan 4.

It can be seen that the Flat Encoding uses twice as many variables as the Compact Tree Encoding for the same number of universal quantifiers. Also, by expanding the universal quantifiers in both Flat and Compact Tree Encoding, we get propositional formulae, in which the latter has twice as many transition relations.

3.4 Leaf-based Encodings

In Section 2.1.1 several different SAT-based encodings were described, and it was noted that it is possible to use these encodings when translating to QBF; the CTE or Flat Encoding can be used with many different state descriptions and constraints. In this section two new approaches will be described, applicable to the CTE framework, that are not possible to apply to translations of Planning to SAT. The technique relies upon the expressivity of the QBF, and specifically, the tree-like structure of the CTE. The resulting formulae contain fewer variables and clauses than do other state-based representation strategies.

The tree-structure of the CTE comes from the expansion of the universal quantifiers. Each level in the tree corresponds to a set of existentially quantified variables, and represents 2^k states, where k is the depth of the level. A plan corresponds to a traversal of the tree. In the encoding of state described in Section 2.1.1 the actions and the fluents were present in every layer; unless they were compiled away. These ideas are illustrated in Figure 3.4.

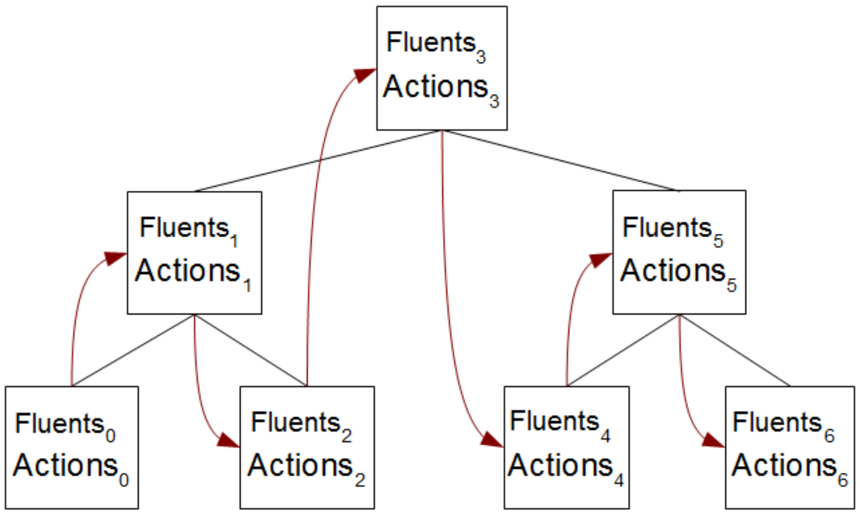


Figure 3.4: The tree formed by expanding universal quantifiers in the CTE with a generic state-based representation. Each level corresponds to a set of existentially quantified variables; each node represents a unique combination of context and variable set. Transitions between states are noted with arrows.

The intuition behind the new approach is to load all of the actions, or all of the fluents, into the innermost quantified layer. The result is that the leaf nodes represent a larger portion of the plan, and since these are the variables that appear under the largest number of contexts the same number of variables capture a greater amount of information. Depending upon which set of variables is moved to the innermost layer we call this representation the Action-leaf or the Fluent-leaf CTE.

We will describe the Action-leaf CTE in detail. The Fluent-leaf CTE is almost identical in construction, with the fluents moved in place of the actions, with one important distinction that will be described later.

The quantification layer of the Action-leaf CTE is:

$$\exists F_k \forall y_k \ldots \exists F_1 \forall y_1 \exists A_0, F, A_1$$

where F_i and A_i are copies of the set of fluents and actions, respectively. This differs from the quantification of the generic CTE described in section 3.2 in which the existential sets represent a whole state: $F \bigcup A$.

The transition constraints τ are also modified. In the general CTE $\tau(X, X')$ enforces action effects and the frame axioms. In the Action-leaf encoding half of the transitions enforce action effects and frame axioms ($\tau_e(F, A, F')$), and the other half enforce action preconditions and frame axioms ($\tau_p(F, A, F')$). Similarly, the state constraints are modified. Only those variables in the innermost quantified set are constrained. These constraints, $\sigma_a(A, F, A')$, enforce the precondition constraints of A', effects of A, and other state constraints. σ_a, τ_p, and τ_e are described in more detail along with an example below.

The resulting formula is

$$I(X_I) \wedge Q^a_k(X_I, X_G) \wedge G(X_G) \tag{3.3}$$

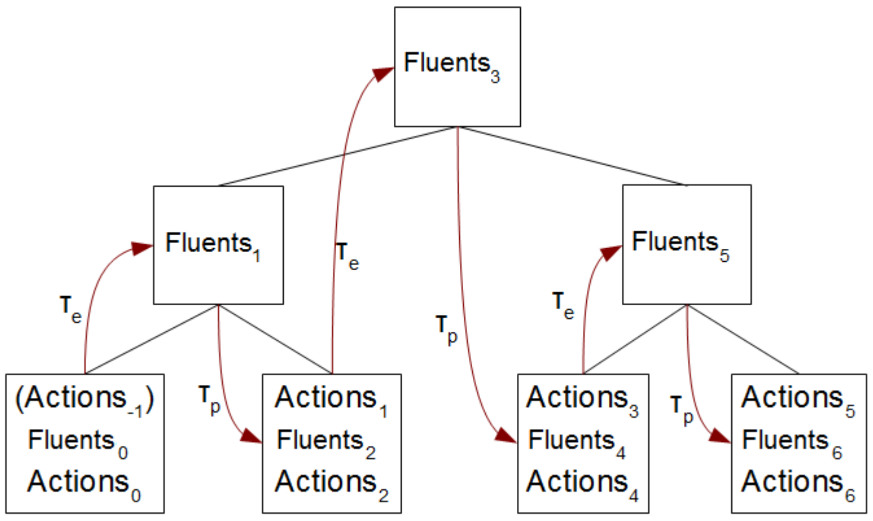
where

$$\begin{array}{l}
Q^a_k(X_I, X_G) := \exists F_k \forall y_k \ldots \exists F_1 \forall y_1 \exists A_0, F, A_1(\\
\quad \sigma(A_0, F, A_1) \\
\quad \wedge \\
\quad ((\bigwedge_{i=1}^{k} \neg y_i) \Rightarrow (X_I \leftrightarrow F \bigcup A_1)) \\
\quad \wedge \\
\quad ((\bigwedge_{i=1}^{k} y_i) \Rightarrow (X_G \leftrightarrow F \bigcup A_1)) \\
\quad \wedge \\
\quad \bigwedge_{i=1}^{k}(\quad ((\neg y_i \wedge \bigwedge_{j=1}^{i-1} y_j) \Rightarrow \tau_e(F, A_1, F_i)) \\
\qquad\qquad \wedge \\
\qquad\qquad ((y_i \wedge \bigwedge_{j=1}^{i-1} \neg y_j) \Rightarrow \tau_p(F_i, A_0, F)) \quad))
\end{array}$$

and is represented by Figure 3.5.

3.5 Example of the Action-leaf CTE

Recall the simple example in Section 1.1.5 on Planning as Satisfiability. The Planning problem is described in Figure 1.1 and Figure 1.2. Described in this subsection is an example of encoding the same problem as a QBF, using the Action-leaf CTE.



Fluents$_3$
Fluents$_1$
Fluents$_5$
T_e
T_e
T_e
T_p
T_p
T_p
(Actions$_{-1}$) Fluents$_0$ Actions$_0$
Actions$_1$ Fluents$_2$ Actions$_2$
Actions$_3$ Fluents$_4$ Actions$_4$
Actions$_5$ Fluents$_6$ Actions$_6$

Figure 3.5: The tree formed by expanding universal quantifiers in the Action-leaf CTE. Each level corresponds to a set of existentially quantified variables; each node represents a unique combination of context and variable set. Transitions between states are noted with arrows.

After grounding we obtain two sets, F and A:

$$\begin{gathered}F := \{(on\,A\,A), (on\,A\,B), (on\,A\,C), (on\,B\,A), (on\,B\,B),\\ (on\,B\,C), (on\,C\,A), (on\,C\,B), (on\,C\,C),\\ (ontable\,A), (ontable\,B), (ontable\,C),\\ (clear\,A), (clear\,B), (clear\,C),\\ (holding\,A), (holding\,B), (holding\,C),\\ (handempty)\}\end{gathered}$$

and

$$\begin{gathered}A := \{(stack\,A\,A), (stack\,A\,B), (stack\,A\,C), (stack\,B\,A), (stack\,B\,B),\\ (stack\,B\,C), (stack\,C\,A), (stack\,C\,B), (stack\,C\,C),\\ (unstack\,A\,A), (unstack\,A\,B), (unstack\,A\,C), (unstack\,B\,A), (unstack\,B\,B),\\ (unstack\,B\,C), (unstack\,C\,A), (unstack\,C\,B), (unstack\,C\,C),\\ (put\text{-}down\,A), (put\text{-}down\,B), (put\text{-}down\,C),\\ (pick\text{-}up\,A), (pick\text{-}up\,B), (pick\text{-}up\,C)\}.\end{gathered}$$

We construct the formula 3.3 for a sufficiently large k, in which:

1. $I(X) \models F_0$ and $I(X) \models \neg f, \forall f \notin F_0$ where

$$F_0 : \left\{ \begin{array}{c}(on\,A\,C), (ontable\,B), (ontable\,C),\\ (clear\,A), (clear\,B), (handempty)\end{array} \right\}$$

2. $G(X) \models F_G$ and $G(X) \models \neg f, \forall f \notin F_G$ where

$$F_G := \{(on\,A\,B), (on\,B\,C)\}$$

3. $\sigma_a(A, F, A')$ models

   - mutual exclusion relations in both A and A'.

     $$\sigma_a(A, F, A') \models \neg a_1 \vee \neg a_2$$

     $\forall a_1, a_2 \in A$ and $\forall a_1, a_2 \in A'$ such that a_1 and a_2 are mutually exclusive. For example:

     $$\begin{array}{l} (\textit{pick-up}\, A) \rightarrow \neg(\textit{pick-up}\, B) \\ (\textit{pick-up}\, A)' \rightarrow \neg(\textit{pick-up}\, B)' \end{array}$$

   - action effects between A and F.

     $$\sigma_a(A, F, A') \models a \rightarrow e$$

     $\forall a \in A$, $e \in F$ with $e \in E_a$. For example:

     $$\begin{array}{l} (\textit{pick-up}\, A) \rightarrow \\ (\textit{holding}\, A) \wedge \neg(\textit{ontable}\, A) \wedge \neg(\textit{clear}\, A) \wedge \neg(\textit{handempty}) \end{array}$$

   - precondition relations between F and A'.

     $$\sigma_a(A, F, A') \models a \rightarrow p$$

     $\forall a \in A'$, $p \in F$ with $p \in P_a$. For example:

     $$(\textit{unstack}\, A\, C)' \rightarrow (\textit{on}\, A\, C) \wedge (\textit{clear}\, A) \wedge (\textit{handempty})$$

4. $\tau_e(F, A, F')$ ensures

   - the effects of an action applied in A are present in F'

     $$\tau_e(F, A, F') \models a \rightarrow e$$

     $\forall a \in A$, $e \in F'$ with $e \in E_a$. For example:

     $$\begin{array}{l} (\textit{pick-up}\, A) \rightarrow \\ (\textit{holding}\, A)' \wedge \neg(\textit{ontable}\, A)' \wedge \neg(\textit{clear}\, A)' \wedge \neg(\textit{handempty})' \end{array}$$

   - that a fluent made true in F' implies a supporting action, or fluent in F

     $$\tau_e(F, A, F') \models f_\beta \rightarrow (f_\alpha \vee A_f)$$

     $\forall f_\beta \in F'$ with corresponding fluent $f_\alpha \in F$. $A_f \subseteq A$ represents the achievers of f_β, that is, all actions a such that $f \in E_a$. For example:

     $$\begin{array}{l} (\textit{holding}\, A)' \rightarrow (\textit{holding}\, A) \vee (\textit{pick-up}\, A) \\ \vee (\textit{unstack}\, A\, A) \vee (\textit{unstack}\, A\, B) \vee (\textit{unstack}\, A\, C) \end{array}$$

5. $\tau_p(F, A, F')$ ensures

   - that the preconditions of an action applied in A are present in F

     $$\tau_p(F, A, F') \models a \rightarrow p$$

     $\forall a \in A$, $p \in F$ with $p \in P_a$ For example:

     $$(\textit{unstack}\, A\, C) \rightarrow (\textit{on}\, A\, C) \wedge (\textit{clear}\, A) \wedge (\textit{handempty})$$

- that a fluent made true in F' implies a supporting action, or fluent in F. This is identical to the description provided for τ_e.

Comparing the Action-leaf CTE (Q_a) to a CTE (Q) that uses a regular state-based representation, we find that there are less variables and less clauses. The number of existential variables in the Q is $(k+1)(|F|+|A|)$, and with Q_a is only $(k+1)|F|+2|A|$. The number of universal variables, and maximum makespan remain the same.

The number of clauses is similarly reduced. Consider the constraints broken down into the following components: precondition P, effect E, mutual exclusion M, and frame axiom R. In Q there are

$$(k+1)(|P|+|M|)+2k(|R|+|E|)$$

clauses; where $|M|$ represents the number of clauses required to enforce the mutual exclusion relationships, and similarly for the the other components. In Q_a there are only

$$2|M|+(k+1)(|E|+|P|)+2k|R|$$

clauses. Using the Action-leaf CTE eliminates $(k-1)(|M|+|E|)$ clauses, assuming $k>0$. Table 3.1 illustrates this reduction, listing the number of variables and clauses in various encodings of Planning benchmark problems. The benchmarks are introduced more properly in Chapter 5. The time-step bounds on the encodings shown in this table were provided by the fix point of the plan graph, as described in Section 2.1.1. This bound is often much lower than the optimal makespan of the solution, in practice the size difference of encodings produced while planning is much greater. As the time-step bound grows so too does k, and as stated the size difference grows linearly in k.

The solution times for the Action-leaf CTE and regular state-based CTE are compared in Chapter 5.

The Fluent-leaf CTE is built in a similar fashion to the Action-leaf CTE, with the positions of the action and fluent sets swapped. Each non-leaf node represents a set of actions, each leaf node represents two sets of fluents and a set of actions. In addition to this, each action set in a non-leaf node must also contain a set of noop actions. This is a major difference between the two Leaf-based state representations.

These new variables must be included because of the frame axiom constraints. The constraints that model the frame axioms involve variables in adjacent sets of fluents, that is, two sets of fluents variables that represent states connected by a transition relation. In the Action-leaf CTE any two sets of fluents that are adjacent are in leaf and non-leaf nodes of the tree. Both τ_e and τ_p model the frame axioms. In the Fluent-leaf representation two sets of fluents that are adjacent are either both in the same leaf node, or in different leaf nodes.

The frame axiom constraints between fluent sets in a single leaf node can be included once, in the same manner as the action mutual exclusion constraints in the Action-leaf representation.

The frame axioms between different leaf nodes are more problematic, and additional variables – the noop action variables – are required to model these constraints.

The Fluent-leaf CTE (Q_f) therefore requires:

$$(k+1)|A| + (k+2)|F|$$

variables, and

$$(k+1)(|P| + |M| + |E| + |R|) + 2|F|$$

clauses. The additional $2|F|$ clauses correspond to the precondition and effect constraints of the new noop actions. Considering $|F| = |R|$ when using explanatory frame axioms, it is clear that the Q_f generally contains fewer clauses than Q, at the cost of $|F|$ more variables.

	CTE		Action-leaf CTE	
problem	variables	clauses	variables	clauses
depots01	362	5777	380	4378
depots02	818	31267	792	21442
depots09	9467	2662347	6631	1376475
depots10	3038	369516	2620	218347
driverlog01	302	4760	286	2198
driverlog02	437	5533	409	3478
driverlog14	3794	180770	3106	71478
driverlog15	8339	771703	6535	233230
freecell01	545	41420	651	35975
freecell02	3113	938905	2328	502131
freecell03	4661	1945422	3447	1040000
opticaltelegraph01	386	5418	498	4424
opticaltelegraph02	578	10539	746	8220
opticaltelegraph13	2690	172998	3474	119672
opticaltelegraph14	2882	197415	3722	136140
pipesnotankage01	207	4209	281	3511
pipesnotankage02	416	8709	398	6582
pipesnotankage19	8323	2774250	5615	1406993
pipesnotankage20	9675	3753986	6383	1924198
rovers01	296	4662	303	3739
rovers02	163	2243	219	2269
rovers19	9602	2607502	7488	1764391
rovers20	13370	4445202	10354	2983691

Table 3.1: Formula sizes for CTE and Action-leaf CTE; time-step bound provided by the fix point of the plan graph.

Chapter 4

Partially Grounded QBF Encoding

The following formula exploits the expressivity of the QBF problem in order to achieve partial grounding of the Planning instance. This means that not all of the propositions and operators of the domain will be grounded with the instance to form fluents and actions. The resulting formula is decreased in size by a possibly exponential amount. This reduction depends upon the number of objects in the problem instance. Intuitively, by not grounding, we are describing an abstract copy of an object *type* and then using the expansion of the tree to copy these variables for every object. In a domain and problem in which there are a great many objects of one type then a great reduction in variables and clauses may be made. In other domains in which there are few objects of each type no savings may be made at all.

The makespan of the problem will be encoded linearly, as in the SAT translations. It is possible to combinine partially grounded QBF encodings with QBF translations whose makespans are exponentially larger than their number of states, such as the Flat and Compact Tree encodings, but this is left for future work.

The principle idea behind the Partially Grounded QBF (PGQBF) Encoding is illustrated in figures 4.1 and 4.2. The figures show portions of a SAT and PGQBF encoding that correpsond to pigeons and the place action described in the example domain below, in figure 4.3.

Figure 4.1 shows the SAT formulation. In this encoding, each action and each fluent is represented by a unique variable. This means that for two time-steps (with one action step between them) there are $3P$ variables required to represent the fluents and actions for this portion of the problem, where P is the number of pigeons.

Figure 4.2 shows the PGQBF formulation of the same problem. Universally quantified variables are included before the variables that represent the fluents and actions. The quantification layer corresponding to this diagram would be:

$$\begin{gathered}\exists lock_1^{place} \dots lock_m^{place} \\ \forall a_1 \dots a_m \\ \exists placed(pigeon)_i,\, place(pigeon),\, placed(pigeon)_{i+1}\end{gathered}$$

where $2^m = p$. This includes only $2m + 3$ variables.

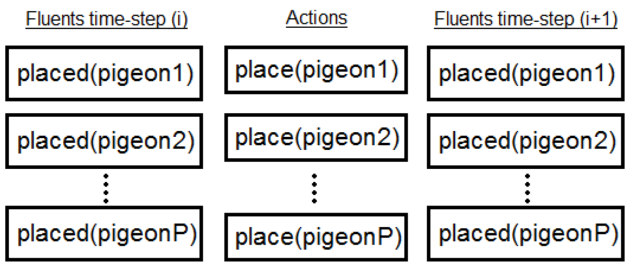


Figure 4.1: Part of a SAT formulation of the pigeonhole problem, with existentially quantified variables represented by square boxes. The state representation is a Graphplan-based encoding with split actions, as described in Section 2.1.

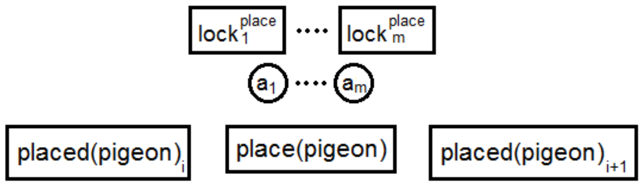


Figure 4.2: Part of a partially grounded QBF formulation of the pigeonhole problem, with existentially quantified variables represented by square boxes and universally quantified variables represented by circles. Expanding the universal quantifiers will produce a tree with $2^m = P$ branches, each corresponding to a unique pigeon.

Upon the expansion of the universal variables, as described in Section 1.2 the fluent and action variables are copied the same number of times they are included in the SAT formulation. The lock variables are used for mutual exclusion between ungrounded actions, and are described in more detail below.

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. Both parts are brought together afterwards in a summary of the constraints as they apply to an example.

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 4.3.

4.1 Partially Grounded QBF Encoding

4.1.1 Splitting propositions and operators

Kautz and Selman [55] used the idea of operator splitting to significantly reduce the size of the resultant encoding. The basic idea is to reduce the arity of

```
(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 (not (empty ?h))
                  (placed ?p))))
```

Figure 4.3: The domain for the pigeonhole problem with operator *place* and propositions *placed*, *empty*, and *in*.

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 [26, 91, 92] and bears some similarity to the alternative state-representation used by Huang et al. [51]. These approaches are described in more detail in Section 2.1.

The encoding presented here uses a split representation similar to that of Kautz and Selman [55] 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.

4.1.2 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 two parts: X^g and X^u, where $X^g := A \bigcup F$ and $X^u := P \bigcup O$.

Additional variables are required 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 is added to X^g:

$$\{lock_0^{o_\alpha}, \ldots, lock_m^{o_\alpha}\} \in X^g.$$

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

$$\{lock_0^{p_\beta}, \ldots, lock_m^{p_\beta}\} \in X^u.$$

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

$$\{lock_0^{p_\alpha}, \ldots, lock_m^{p_\alpha}\} \in X^u.$$

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

$$\begin{aligned}
&X_i^g := \{\\
&\quad place[0]_i,\\
&\quad lock_0^{place[1]_i}, \ldots, lock_m^{place[1]_i},\\
&\quad lock_0^{place[2]_i}, \ldots, lock_m^{place[2]_i}\}\\
&X_i^u := \{\\
&\quad place[0]_i,\ place[1]_i,\\
&\quad in[0]_i,\ in[1]_i,\\
&\quad placed_i,\ empty_i,\\
&\quad lock_0^{in[0]_i}, \ldots, lock_m^{in[0]_i},\\
&\quad lock_0^{in[1]_i}, \ldots, lock_m^{in[1]_i}\}.
\end{aligned}$$

As the number of pigeons or pigeonholes is increased, the size of the lock variable sets grow 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 (PGQBF) 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:

$$\begin{array}{l} \exists X_1^g \ldots X_{n+1}^g \forall a_1 \ldots a_m \exists X_1^u \ldots X_{n+1}^u \cdot (\\ \quad I(X_1^g \cup X_1^u) \wedge G(X_{n+1}^g \cup X_{n+1}^u) \\ \quad \wedge \bigwedge_{i=1}^{n} \tau_{qbf}(X_i^g \cup X_i^u, X_{i+1}^g \cup X_{i+1}^u) \\ \quad \wedge \bigwedge_{i=1}^{n} \sigma_{qbf}(X_i^g \cup X_i^u)). \end{array} \tag{4.1}$$

A *plan for* Φ_n is an interpretation satisfying (4.1).

The state constraints $\sigma_{qbf}(X_i^g \cup X_i^u)$ ensure that $O_i \bigcup A_i$ represents 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.

4.1.3 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. its 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 of 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 both 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:

$$\begin{aligned}&o_\alpha \rightarrow (lock_j^{o_\alpha} \leftrightarrow a_j), \qquad\qquad for\, j = 1, \ldots, m.\\&a_\beta \wedge \textstyle\bigwedge_{j=1}^{m}(lock_j^{o_\alpha} \leftrightarrow a_j) \rightarrow o_\alpha\end{aligned}$$

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

$$place[1](?p) \rightarrow (lock_1^{place[1](?p)} \leftrightarrow a_1)$$

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 $lock_1^{place[1](?p)} \models \bot$.

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

The second constraint:

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

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.

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

Figure 4.4: The operator *remove* for the pigeonhole domain.

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:

$$\begin{aligned}&\mathit{place}[1](?p) \rightarrow \neg \mathit{placed}(?p)\\&\wedge\\&\mathit{place}[2](?h) \rightarrow \mathit{empty}(?h).\end{aligned}$$

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 4.4. 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 *pigeon*1 was in *pigeonhole*1 and *pigeon*2 in *pigeonhole*2 it would be possible to remove *pigeon*1 from *pigeonhole*2. This is avoided by adding the constraint:

$$o_\alpha \rightarrow \bigwedge_{i=1}^{m}(\mathit{lock}_i^{o_\beta} \leftrightarrow \mathit{lock}_i^{p_\beta})$$

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

$$\mathit{lock}_0^{in[2](?h)}, \ldots, \mathit{lock}_m^{in[2](?h)}.$$

The correct pigeonhole means the pigeonhole from which it is being removed, as stored in the operator lock variables

$$\mathit{lock}_0^{remove[2](?h)}, \ldots, \mathit{lock}_m^{remove[2](?h)}.$$

The resulting constraints are:

$$\begin{aligned}&\mathit{remove}[1](?p) \rightarrow\\&\qquad \bigwedge_{i=1}^{m}(\mathit{lock}_i^{remove[2](?h)} \leftrightarrow \mathit{lock}_i^{in[2](?h)})\\&\wedge\\&\mathit{remove}[2](?h) \rightarrow\\&\qquad \bigwedge_{i=1}^{m}(\mathit{lock}_i^{remove[1](?p)} \leftrightarrow \mathit{lock}_i^{in[1](?p)})\end{aligned}$$

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

4.1.4 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. fluents true in X_{i+1} and not added by an action in X_i were true in X_i.

Consider $\tau(X, X')$; in the following description of the constraints we will use v and v' to distinguish between variables belonging to the two sets, where $v \in X$ and $v' \in X'$.

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

$$\begin{array}{l} o_\alpha \rightarrow p'_\alpha \\ a_\alpha \rightarrow f'_\alpha \end{array}$$

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:

$$\begin{array}{l} \mathit{place}[1](?p) \rightarrow \mathit{placed}'(?p) \\ \wedge \\ \mathit{place}[2](?h) \rightarrow \neg \mathit{empty}'(?h). \end{array}$$

Additionally, the proposition lock variables must be set:

$$o_\alpha \rightarrow \bigwedge_{i=1}^{m} (\mathit{lock}_i^{o_\beta} \leftrightarrow \mathit{lock}_i^{p'_\beta})$$

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:

$$\begin{array}{l} \mathit{place}[1](?p) \rightarrow \bigwedge_{i=1}^{m} (\mathit{lock}_i^{\mathit{place}[2](?h)} \leftrightarrow \mathit{lock}_i^{\mathit{in}'[2](?h)}) \\ \wedge \\ \mathit{place}[2](?h) \rightarrow \bigwedge_{i=1}^{m} (\mathit{lock}_i^{\mathit{place}[1](?p)} \leftrightarrow \mathit{lock}_i^{\mathit{in}'[1](?p)}). \end{array}$$

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:

$$\begin{array}{l} p_\alpha \rightarrow p'_\alpha \vee \bigvee D_{p_\alpha} \\ f_\alpha \rightarrow f'_\alpha \vee \bigvee D_{f_\alpha} \\ \neg p_\alpha \rightarrow \neg p'_\alpha \vee \bigvee A_{p_\alpha} \\ \neg f_\alpha \rightarrow \neg f'_\alpha \vee \bigvee A_{f_\alpha} \end{array}$$

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 includes p'_α as an add effect. D_{f_α} and A_{f_α} are similarly defined sets of action fluents.

In the example:

$$\begin{array}{l} \mathit{empty}(?h) \rightarrow \mathit{empty}'(?h) \vee \mathit{place}[2](?h) \\ \wedge \\ \mathit{placed}(?h) \rightarrow \mathit{placed}'(?h) \\ \wedge \\ \neg \mathit{empty}(?h) \rightarrow \neg \mathit{empty}'(?h) \\ \wedge \\ \neg \mathit{placed}(?h) \rightarrow \neg \mathit{placed}'(?h) \vee \mathit{place}[1](?p). \end{array}$$

The locks are maintained using the constraints:

$$p'_\alpha \rightarrow \bigvee A_{p_\alpha} \vee \bigwedge_{i=1}^{m}(lock^{p_\beta} \leftrightarrow lock'^{p_\beta})$$

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

$$\begin{aligned}
&in[1]'(?p) \rightarrow \\
&\qquad (lock_i^{in[2](?h)} \leftrightarrow lock_i'^{in[2](?h)}) \vee place[1](?p) \\
&\wedge \\
&in[2]'(?h) \rightarrow \\
&\qquad (lock_i^{in[1](?p)} \leftrightarrow lock_i'^{in[1](?p)}) \vee place[2](?h).
\end{aligned}$$

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

4.2 Example of the Partially Grounded QBF Encoding

Putting everything together we arrive at the QBF instance Φ_n, constructed according to formula 4.1, 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

$$\begin{aligned}
&X_i^g := \{ \\
&\quad place[0]_i, \\
&\quad lock_0^{place[1]_i}, \dots, lock_m^{place[1]_i}, \ \} \\
&\quad lock_0^{place[2]_i}, \dots, lock_m^{place[2]_i} \} \\
&X_i^u := \{ \\
&\quad place[0]_i,\ place[1]_i, \\
&\quad in[0]_i,\ in[1]_i, \\
&\quad placed_i,\ empty_i, \\
&\quad lock_0^{in[0]_i}, \dots, lock_m^{in[0]_i}, \\
&\quad lock_0^{in[1]_i}, \dots, lock_m^{in[1]_i} \}
\end{aligned}$$

and the constraints are defined by Figure 4.5. 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 $O(m(n+1))$.

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

Chapter 5

Results

5.1 Comparing Encodings with Exponential Time-steps

Chapter 3 presented two encodings with sizes logarithmic in the number of time-steps, the Flat encoding, first presented in Rintanen [80]; and the Compact Tree encoding (CTE), which is novel. We showed that both are semantically equivalent to SAT-based encodings from formula (1.1) presented in Section 1.1.4. This section aims to investigate how these encodings behave in practice. In particular we hypothesise that:

- the CTE is smaller and uses less memory than the Flat encoding (Section 5.1.1);
- the QBF solvers find solutions faster with the CTE, compared to the Flat encoding (Sections 5.1.2, 5.1.3, and 5.1.4);
- the number of quantifier alternations in the encoding is an important factor in this performance improvement (Section 5.1.2);
- the QBF-based encodings are much smaller than SAT-based encodings, and require far less memory to solve (Section 5.1.5);
- there is a performance gap between the QBF-based encodings solved with QDPLL solvers and similar SAT-based encodings (Section 5.1.5); and
- there is a similar performance gap between the QBF-based encodings and the current state-of-the-art SAT-based Planning systems (Section 5.1.5).

It is difficult to estimate the amount of time required by a solver to solve a QBF instance, and the results vary depending upon the solver used. In this section the solver characteristics described in Section 2.3 are used to explain the performance of various QBF-based encodings and experimental results are provided to support the explanation. We use four different solvers for the main experiments: QuBE7.0 [41, 64]; DepQBF (v0.1) [61]; CirQit2.1 [44]; and Quantor (v3.0) [4].

QBF-based encodings that are logarithmic in the number of time-steps are not yet competitive with similar SAT-based encodings. Although semantically equivalent to SAT encodings, the QBF representation is much more difficult to solve.

The problems from the small hard track of the QBF Evaluation in 2010 (QBFEval'10) [38] have up to ~2000 variables, a similar size to many encodings of Planning instances. However, the encodings presented in Chapter 3 do not benefit from any preprocessing, such that as performed by sQUEEZEBF [40], the preprocessor for QUBE7.0.

5.1.1 Comparing the size of CTE and Flat encodings

When comparing the size of CTE and Flat encodings of the same Planning problem there are three aspects to consider:

- the number of variables and clauses;
- the memory footprint during solving; and
- the number of quantifier alternations.

As demonstrated in Section 3.3 there will be fewer variables and alternations in the CTE encoding than in the Flat encoding. This can be shown as follows. Assume that both encodings, which are state-based encodings, use the same variable set X to represent the state. This can be an actions-only, SAS$^+$-based, Graphplan-based or any other state-based representation. To represent a plan of n time-steps the CTE requires $log_2(n+1)|X|$ variables where $|X|$ is the number of variables in X. These states are arranged in $log_2(n+1) - 1$ alternations. In comparison, the Flat encoding requires $(3log_2(n)+2)|X|$ existential variables arranged in $log_2(n)$ alternations. The sizes of encodings of various Planning problems from the International Planning Competition (IPC) are shown in Table 5.2. The number of alternations present in encodings of different makespans is shown in Table 5.1.

As shown in Table 5.2 the CTE contains a smaller number of variables, but many more clauses, resulting in a larger formula. The encodings presented in Table 5.2 were solved using a variety of solvers. The experiments were run to illustrate that:

- the larger formula size is negligible when compared to the memory use during solving; and
- the CTE often uses less memory than the Flat encoding despite the larger formula size.

The encodings generated for Table 5.2 were passed to the QDPLL-based solvers QUBE7.0 and DEPQBF as well as the resolution solver QUANTOR. The solvers were run on machines with 8GB of memory and the amount of memory used by the solver was recorded at small time intervals during the solution process. The results are presented in two ways: *maximum* and *average*. The average memory is the mean of all readings giving a more accurate approximation of the average memory used over time. The results are shown in Tables 5.3, 5.4, and 5.5. Figures 5.1, 5.2, and 5.3 chart the average memory use shown in the tables.

Quantifier Alternations								
Makespan	2	3	4	5	6	7	8	9
CTE	1	1	2	2	2	2	3	3
Flat	1	2	2	3	3	3	3	4

Table 5.1: Number of quantifier alternations for various makespans.

	CTE		Flat	
Problem	variables	clauses	variables	clauses
depots01	362	5777	1323	4271
depots03	1979	131391	5437	42125
depots10	3038	369516	11135	124280
depots16	4922	818214	18043	277408
driverlog01	302	4760	1103	3134
driverlog03	482	6517	1763	5171
driverlog13	3557	260072	13038	50606
driverlog15	8339	771703	30572	172653
freecell01	545	41420	2178	21062
freecell03	4661	1945422	17086	542587
gripper01	41	243	162	420
gripper03	89	895	354	1036
gripper08	209	3925	834	2996
gripper10	257	5697	1026	3948
opticaltelegraph01	386	5418	1411	4262
opticaltelegraph03	770	17268	2819	10012
opticaltelegraph12	2498	150189	9155	54301
opticaltelegraph14	2882	197415	10563	68235
philosophers01	212	1730	773	2130
philosophers03	422	4788	1543	4580
philosophers18	1997	70053	7318	33155
philosophers20	2207	84399	8088	38325
pipesnotankage01	207	4209	826	2917
pipesnotankage03	770	28724	2050	10153
pipesnotankage18	7655	2304186	26786	632155
pipesnotankage20	9675	3753986	33856	1008796
rovers01	296	4662	786	2839
rovers03	362	5058	1323	4460
rovers18	6623	687668	24280	279466
rovers20	13370	4445202	49019	1576696
tpp01	47	187	168	405
tpp03	113	491	410	1005
tpp18	22483	1323829	61823	414921
tpp20	27191	1470445	95162	560925
zeno01	26	261	52	190
zeno03	968	41157	2578	18539
zeno12	4907	611491	13082	218841
zeno14	20567	6208928	54842	2097420

Table 5.2: Number of variables and clauses for encodings of various problems in the IPC benchmark suite. The time-step bound for each encoding is provided by the fix point of the plan graph.

	DepQBF			
	average		maximum	
Problem	CTE	Flat	CTE	Flat
depot02	97092	**89877**	**144832**	177116
driverlog02	**91113**	159937	**139772**	250604
driverlog04	**11139**	13097	**14888**	129312
driverlog05	**19232**	82021	**31292**	173980
driverlog06	**4354**	102236	**5208**	19512
driverlog07	**4505**	119064	**6192**	185668
driverlog10	**79380**	178593	**155660**	306380
gripper04	**2132**	7706	**2132**	10808
pipesnotankage02	**29705**	67511	**38760**	112840
pipesnotankage03	**35776**	150743	**50700**	285556
pipesnotankage04	**51588**	195679	**79796**	347636
rovers05	**7811**	22868	**10144**	46364
rovers06	**118970**	341414	**164552**	502552
rovers07	**6970**	38803	**9640**	73248
rovers09	**118277**	444591	**179360**	857304
rovers12	**41547**	344861	**59284**	559468

Table 5.3: Memory used solving problems using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with DepQBF; sizes to the nearest kb.

	QuBE7.0			
	average		maximum	
Problem	CTE	Flat	CTE	Flat
depots02	523978	**338500**	817140	**572568**
driverlog04	**58792**	188588	**76412**	263980
driverlog05	214205	**149449**	304140	**230936**
driverlog06	**42580**	45334	**43060**	51932
driverlog07	**43815**	130147	**47576**	222808
driverlog08	**76121**	139567	**136280**	226228
freecell01	753847	**327463**	1138380	**503608**
gripper04	**39192**	42781	**39192**	45792
pipesnotankage02	**55235**	495809	**66164**	721212
pipesnotankage03	**66856**	497625	**88236**	840516
pipesnotankage04	**118570**	371777	**209516**	564488
rover05	**54363**	54501	**65964**	77964
rover06	**96410**	365736	**156948**	693828
rover07	**44292**	53918	**46940**	70168

Table 5.4: Memory used solving problems using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with QuBE7.0; sizes to the nearest MB.

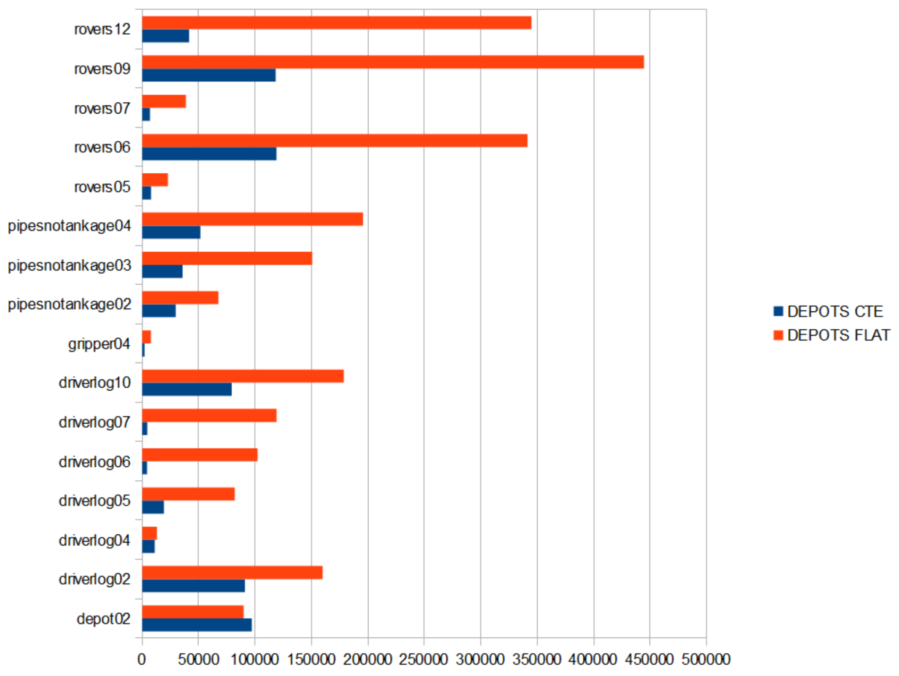


Figure 5.1: Average memory use for problems solved using the CTE against Flat encoding; solving QBFs with DepQBF; memory in kb.

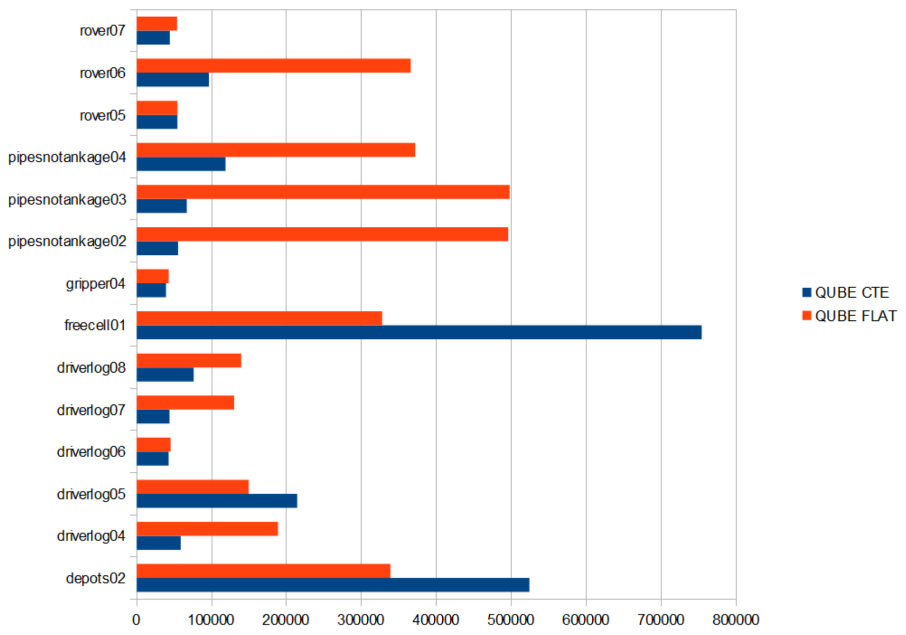


Figure 5.2: Average memory use for problems solved using the CTE against Flat encoding; solving QBFs with QuBE7.0; memory in kb.

	Quantor			
	average		maximum	
Problem	CTE	Flat	CTE	Flat
depots02	24630	**13292**	33076	**13292**
depots03	**71780**	82220	**107876**	121080
depots08	**466702**	522555	**611224**	677996
depots10	**224821**	246330	**407468**	466608
depots13	**258005**	295838	**528024**	587964
depots16	333407	**283102**	748700	**500336**
driverlog09	**21901**	27742	**25440**	28320
driverlog10	**17240**	19987	**17240**	22496
driverlog11	**33428**	49033	**48032**	53208
freecell01	**75149**	88082	**133996**	139524
freecell02	498139	**433331**	999668	**582148**
freecell03	**761051**	821409	1099752	**1063204**
gripper05	**4436**	4483	**4436**	4828
pipesnotankage05	**16243**	28636	**26300**	28636
pipesnotankage06	**21530**	28884	**26484**	28884
pipesnotankage07	**25449**	33351	**46596**	47320
pipesnotankage08	**29140**	43968	**46596**	47584
pipesnotankage09	83248	**63018**	117868	**66984**
pipesnotankage11	**320087**	360006	**470132**	521272
pipesnotankage13	**488553**	525060	**707672**	783816
rovers06	**15382**	17095	**21788**	22784
rovers07	**4736**	6092	**4736**	6092
rovers09	**32626**	32767	**48900**	55396
rovers11	75235	**75104**	**117248**	123764

Table 5.5: Memory used solving problems using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with Quantor; sizes to the nearest MB.

As can be seen from tables 5.3, 5.4, and 5.5 the CTE usually requires far less memory than the Flat Encoding in order to solve Planning problems. This can be partly ascribed to the number of alternations, and so states, that are required. As described in Section 3.3 the CTE requires one less alternation than the Flat Encoding for certain makespans. Most problems will pass through one or more of these makespans during the process of iterative deepening. The QBF instances generated at this point for the Flat Encoding contain an additional state, comprising $3|X|$ existential variables and accompanying clauses.

That the formula size is negligible when comparing the memory use of the two QBF encodings is illustrated by *pipesnotankage*03. Despite the CTE containing almost three times as many clauses as the Flat Encoding, the average amount of memory used when solving with the solver DepQBF is almost four times lower. When solving with QuBE7.0 the CTE requires roughly an eighth of the memory required by the Flat Encoding.

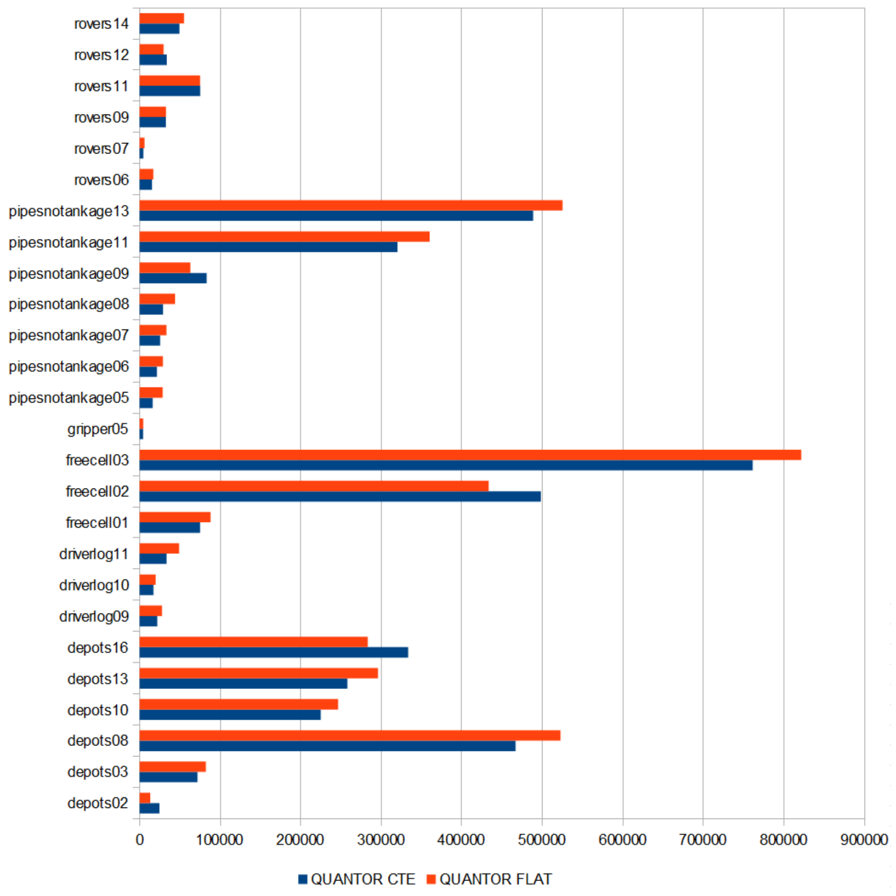


Figure 5.3: Average memory use for problems solved using the CTE against Flat encoding; solving QBFs with QUANTOR; memory in kb.

5.1.2 Timing CTE and Flat encodings with DPLL solvers

When using search-based (QDPLL) QBF solvers the number of alternations is an important factor in the time to solve. QDPLL-based solvers suffer from the restricted selection of the variable upon which to branch. As described in Section 2.3.1 the variable selection phase is limited to variables in the outermost quantified set. For both the CTE and the Flat encoding this represents the midpoint of the plan. These variables are as far as possible from the tightest constraints: those representing the initial state and the goal state. Due to this restricted selection the solver must make a guess at the midpoint of the plan, only considering the state constraints, σ. Upon recursion the midpoint must again be guessed at, until the innermost existential layer is reached, at this point τ, I and G are considered and pruning performed. This procedure involves a large amount of backtracking and is very inefficient.

Experiments were run to determine whether:

- the CTE performs better than the Flat encoding on both QDPLL-based

Solver	CTE	Flat
QuBE7.0	23	0
DepQBF	9	0
Quantor	3	1

Table 5.6: Number of Planning problems solved by the CTE or Flat encoding that were not solved by the other.

solvers; and

- QDPLL-based solvers still perform poorly on both encodings, in terms of the number of backtracks.

The time taken to encode and solve various Planning problems was recorded for QDPLL-based solvers QuBE7.0 and DepQBF. The problems were solved on a machine with 8GB of memory and a 3.2GHz processor, using the iterative deepening technique described in Figure 2.10. The solvers were given a time limit of 4 hours per encoding. Total times were recorded for solving over all time-step bounds. These times are shown in Tables 5.7 and 5.8 for QuBE7.0 and DepQBF respectively. Figures 5.4 and 5.5 give a clear comparison of these times. Table 5.6 shows the number of problems that were solved using one encoding and could not be solved using the other.

In Figures 5.4 and 5.5 any data points above the line correspond to problems from the benchmark set that were solved faster using the CTE. From the results it can be seen that the CTE outperforms the Flat encoding on every problem domain tested and using both solvers, for many problems finding a makespan optimal solution an order of magnitude faster. This time improvement scales with the size of the problem, producing a series of points in the figures parallel to the central diagonal. This is to be expected as both encodings provide a logarithmic reduction in the number of time-steps, and scale similarly as a result.

The CTE outperforms the Flat encoding largely due to the smaller number of quantifier alternations for most makespans. Table 5.1 shows the number of universal variables present in encodings of Planning problems with various makespans. As shown in Section 3.3 the CTE has one fewer alternation than the Flat encoding, with the exception of makespans 2^k where $k \in \mathbb{Z}$.

Encodings with fewer alternations are much easier to solve. To illustrate this Figures 5.6 and 5.7 show the ratio of solution times for CTE vs. Flat encoding for encodings of the benchmark problems at various makespans. Comparing Table 5.1 with Figures 5.6 and 5.7 reveals that the CTE solves problems much faster than the Flat encoding on any encoding of a makespan for which the number of alternations differ. On makespans four and eight, although the CTE still dominates, the ratio is more even. This corresponds exactly with the plan lengths at which the number of alternations is equal.

In practice most problems have optimal makespans larger than the lower bound provided by the Plan Graph. As a result several encodings must be solved for every problem, if the iterative deepening strategy described by 2.10 is being used. Supposing a lower bound $n + 1$ is determined for a problem, and the problem has an optimal makespan of $m > n$. Using the iterative deepening strategy $m - n$ encodings are generated, which are likely to encompass levels at which the CTE contains fewer quantifier alternations as does the Flat encoding.

QuBE7.0					
Problem	CTE	Flat	Problem	CTE	Flat
depots01	760	7072	*philosophers18*	3252524	-
depots02	1005356	-	*philosophers19*	1760582	-
driverlog01	375	1227	*philosophers20*	1669300	-
driverlog03	1529	17787	*pipesnotankage01*	85	675
driverlog04	33175	-	*pipesnotankage02*	43159	-
driverlog05	1207393	724462	*pipesnotankage03*	35703	-
driverlog06	1352	22122	*pipesnotankage04*	466321	-
driverlog07	10934	459226	*rovers01*	763	2718
driverlog08	199054	-	*rovers02*	100	444
freecell01	1924055	-	*rovers03*	1059	5324
gripper01	44	41	*rovers04*	182	812
gripper02	42	92	*rovers05*	34595	80103
gripper03	692	2846	*rovers06*	643805	-
gripper04	1132	9328	*rovers07*	13564	133544
opticaltelegraph01	1110578	-	*tpp01*	77	199
philosophers01	3964	17450	*tpp02*	75	243
philosophers02	17629	64621	*tpp03*	83	280
philosophers03	30665	132153	*tpp04*	89	294
philosophers04	102270	292926	*tpp05*	732	2096
philosophers05	246870	472545	*tpp06*	29547	115144
philosophers06	401424	888691	*tpp07*	48000	123493
philosophers07	504276	1056309	*tpp08*	127773	284190
philosophers08	504864	1410098	*tpp12*	39	-
philosophers09	962146	1195237	*zeno01*	19	61
philosophers10	921106	-	*zeno02*	955	35577
philosophers11	205713	2234704	*zeno03*	9704	479426
philosophers12	1021774	-	*zeno04*	6548	250395
philosophers13	1667595	-	*zeno05*	8156	-
philosophers14	1805404	-	*zeno06*	1001120	-
philosophers15	2320674	-	*zeno07*	62754	-
philosophers16	2478535	-	*zeno08*	835653	-

Table 5.7: Time taken to solve instances using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with QuBE7.0; times in ms. “-” means the encoding ran out of time.

DepQBF					
Problem	CTE	Flat	Problem	CTE	Flat
depots01	140	643	*philosophers16*	565841	1770275
depots02	101095	573636	*philosophers17*	576066	2111995
driverlog01	68	310	*philosophers18*	753490	2258978
driverlog02	1224555	-	*philosophers19*	743177	2338140
driverlog03	932	12998	*philosophers20*	964156	2990766
driverlog04	35319	1192730	*pipesnotankage01*	60	95
driverlog05	323176	960710	*pipesnotankage02*	58334	434578
driverlog06	1195	7755	*pipesnotankage03*	55743	544104
driverlog07	7499	276473	*pipesnotankage04*	284712	834895
driverlog08	71619	-	*rovers01*	181	788
driverlog10	821889	-	*rovers02*	48	106
freecell01	513071	-	*rovers03*	552	2172
gripper01	19	22	*rovers04*	73	179
gripper02	14	18	*rovers05*	3599	34092
gripper03	395	1532	*rovers06*	1084477	-
gripper04	762	3891	*rovers07*	7886	43319
opticaltelegraph01	186104	907674	*rovers09*	1261677	2085207
opticaltelegraph02	799036	2755832	*rovers12*	146931	-
opticaltelegraph03	2240081	-	*tpp01*	16	17
philosophers01	2232	7025	*tpp02*	11	18
philosophers02	5006	16236	*tpp03*	13	28
philosophers03	9817	37327	*tpp04*	16	38
philosophers04	20622	56363	*tpp05*	204	901
philosophers05	31684	103403	*tpp06*	10040	29347
philosophers06	47097	133095	*tpp07*	19060	55662
philosophers07	49323	234373	*tpp08*	23693	82967
philosophers08	69093	315842	*zeno01*	91	37
philosophers09	108724	412922	*zeno02*	1693	4744
philosophers10	124423	506484	*zeno03*	14505	82659
philosophers11	158224	699470	*zeno04*	9851	79918
philosophers12	173314	868314	*zeno05*	54010	298783
philosophers13	234561	1241162	*zeno06*	368960	-
philosophers14	265193	1309122	*zeno07*	280907	941122
philosophers15	393328	1665619	*zeno08*	964849	-

Table 5.8: Time taken to solve instances using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with DepQBF; times in ms.

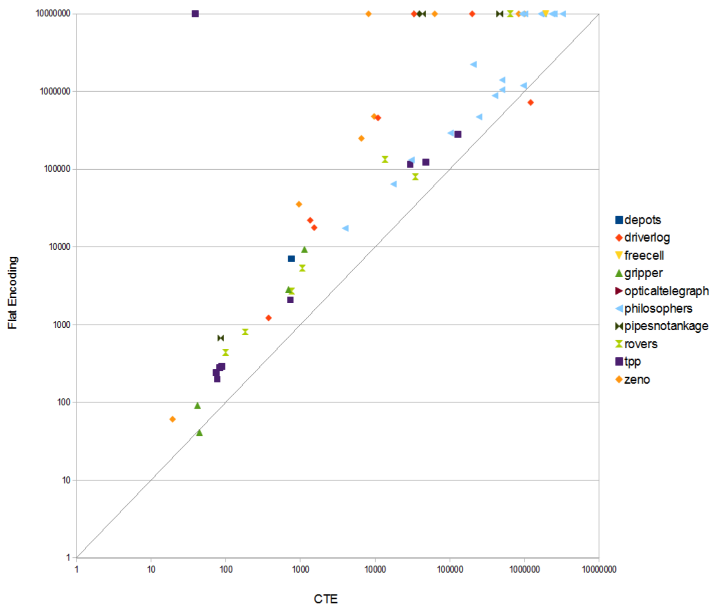


Figure 5.4: Times on problems solved using the CTE against Flat encoding; solving QBFs with QuBE7.0; times in ms.

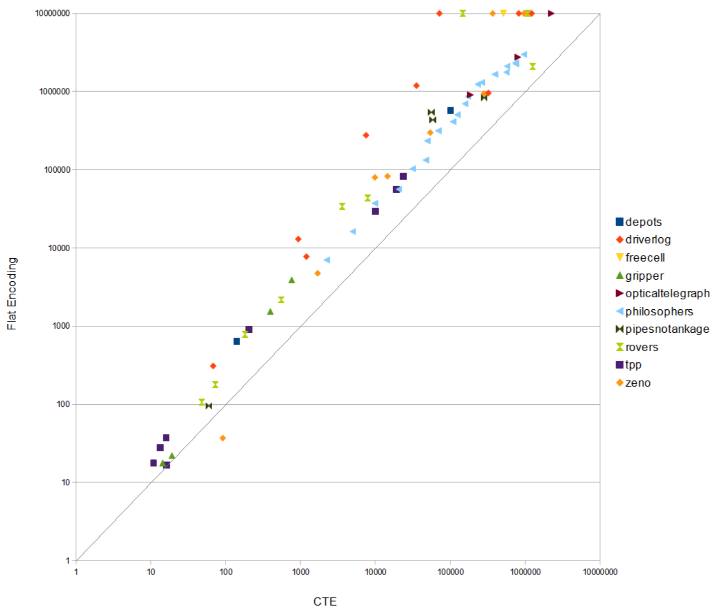


Figure 5.5: Times on problems solved using the CTE against Flat encoding; solving QBFs with DepQBF; times in ms.

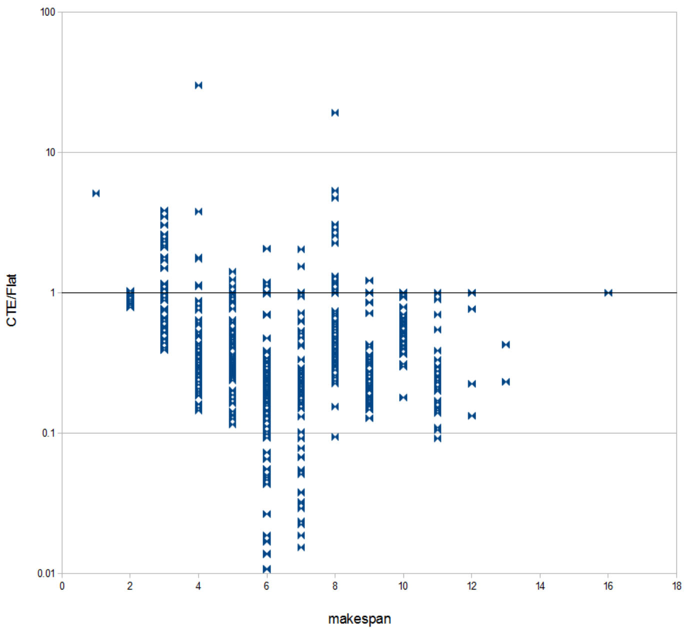


Figure 5.6: Ratio of solution times, (CTE/Flat) on encodings with various makespans; solving QBFs with DepQBF.

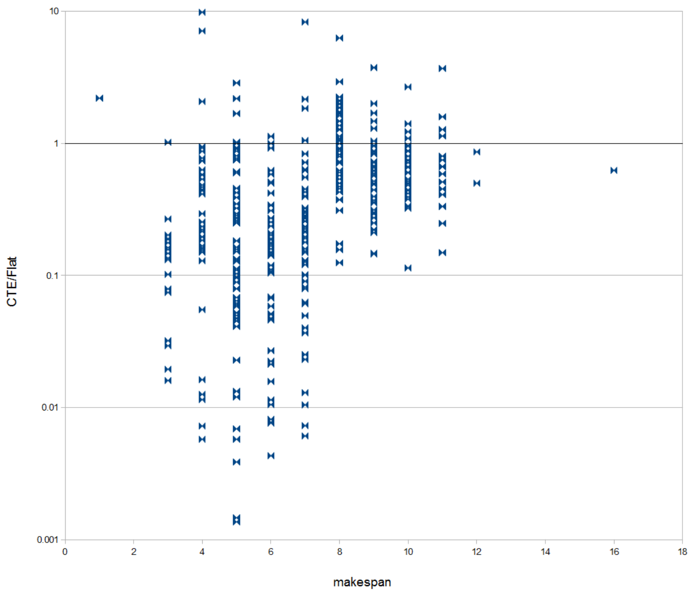


Figure 5.7: Ratio of solution times, (CTE/Flat) on encodings with various makespans; solving QBFs with QuBE7.0.

Problem	CirQit	DepQBF(CTE)
depots01	374279	140
driverlog01	422526	68
driverlog03	1964828	932
gripper01	30	19
gripper02	48	14
gripper03	33351	395
gripper04	153140	762
pipesnotankage01	2048	60
rovers01	104075	181
rovers02	3083	48
rovers03	488406	552
rovers04	137015	73
tpp01	29	16
tpp02	44	11
tpp03	88	13
tpp04	76	16
tpp05	9514	204
zeno01	179	91
zeno02	864750	1693

Table 5.9: Time taken to solve some instances using the ISCAS-85 translation of the CTE, solving QBFs with CirQit2.1; times in ms.

5.1.3 Timing CTE and Flat encodings with other solvers

The same experiments were run using non QDPLL-based solvers, Quantor [4] and CirQit2.1 [44]. CirQit2.1 uses the non-CNF representation format ISCAS-85, as described in Section 2.3. The circuit specification does not represent axioms in the same way as a clausal QBF and the redundant variables in the Flat encoding can be removed without adding new constraints. The resulting ISCAS-85 encoding is identical to the CTE translation to ISCAS-85. For this reason only one set of encodings were solved by CirQit2.1. The time taken to solve the problems is displayed in Tables 5.9 and 5.10.

From the tables it can be seen that CirQit2.1 performs very poorly on this kind of problem, only solving a fraction of the problems in the benchmark set. Quantor performs well, solving more problems than either QDPLL-based solver. Quantor resolves much of the problem to SAT, resulting in faster times at the expense of memory use. The increased memory can be seen by comparing Table 5.5 to Tables 5.3 and 5.4. However, this is still a smaller memory footprint than that used by SAT, as described below.

Quantor					
Problem	CTE	Flat	Problem	CTE	Flat
depots01	166	201	*philosophers14*	16821	13015
depots02	2148	1765	*philosophers15*	20714	15763
depots03	49763	38796	*philosophers16*	25293	19541
depots07	54610	39759	*philosophers17*	30885	24297
depots08	1518408	1242955	*philosophers18*	37052	29431
depots10	265283	236035	*philosophers19*	43451	35401
depots13	209085	169113	*philosophers20*	51151	42552
depots16	406147	299706	*pipesnotankage01*	44	59
driverlog01	33	43	*pipesnotankage02*	365	450
driverlog02	645	718	*pipesnotankage03*	772	740
driverlog03	181	233	*pipesnotankage04*	827	785
driverlog04	419	458	*pipesnotankage05*	2049	1924
driverlog07	314	446	*pipesnotankage08*	8067	8024
driverlog08	488	671	*pipesnotankage09*	65654	74406
driverlog09	8206	16208	*pipesnotankage11*	766268	693314
driverlog10	2723	3166	*pipesnotankage13*	1560164	1428158
driverlog11	7206	9065	*rovers01*	176	185
freecell01	36541	26272	*rovers02*	63	76
freecell02	1228708	988268	*rovers03*	357	353
freecell03	1468227	1357045	*rovers04*	92	109
gripper01	21	22	*rovers05*	1121	1079
gripper02	21	40	*rovers06*	8028	7195
gripper03	111	139	*rovers07*	1228	1187
gripper04	133	183	*rovers09*	12568	10952
gripper05	7137	6485	*rovers11*	112884	96757
opticaltelegraph01	2952	2557	*rovers12*	25324	20252
opticaltelegraph02	7964	5883	*rovers14*	34349	26904
opticaltelegraph03	18841	11923	*tpp01*	17	19
opticaltelegraph04	39166	23606	*tpp02*	15	22
opticaltelegraph05	70293	47280	*tpp03*	16	22
opticaltelegraph06	118031	80692	*tpp04*	15	18
opticaltelegraph07	186368	127855	*tpp05*	61	93
opticaltelegraph11	697263	567679	*tpp09*	2555	3070
opticaltelegraph12	897333	739620	*tpp10*	2983	3656
opticaltelegraph13	1169080	955417	*tpp11*	22647	21347
opticaltelegraph14	1456770	1175442	*tpp12*	24244	24358
philosophers01	317	338	*tpp13*	29431	27854
philosophers02	471	545	*tpp18*	314362	-
philosophers03	725	817	*zeno01*	21	-
philosophers04	1037	1204	*zeno02*	678	619
philosophers05	1484	1588	*zeno03*	1523	1428
philosophers06	2014	2211	*zeno04*	1235	1127
philosophers07	2708	2834	*zeno05*	4330	3545
philosophers08	3630	3601	*zeno06*	6703	5713
philosophers11	8342	7448	*zeno09*	60450	50197
philosophers12	10606	8975	*zeno10*	69223	-
philosophers13	13452	10732	*zeno11*	-	171838

Table 5.10: Time taken to solve some instances using the Compact Tree Encoding (CTE) and Flat Encoding, solving QBFs with Quantor; times in ms. Some results are excluded from the table due to space constraints.

Solver	CTE	Action-leaf CTE
QUBE7.0	5	6

Table 5.11: Number of Planning problems solved by the CTE or Action-leaf CTE encoding that were not solved by the other.

5.1.4 Timing CTE using Leaf-based state representations

The experiments were also run using the action-leaf alternative state representation described in Section 3.4. The encodings were solved using QUBE7.0 and the times are shown in Figure 5.8. The figure compares the times with those for solving the CTE using a regular state-based representation.

As can be seen from the figure the action-leaf representation outperforms the regular state-based representation on almost every problem when solving the CTE with QUBE7.0.

Table 5.11 shows the number of problems solved uniquely using the two representations. Although the representations performed better on certain domains, the action-leaf representation performing better overall, the number of solved problems remained similar; the CTE with the action-leaf representation solving only one problem more.

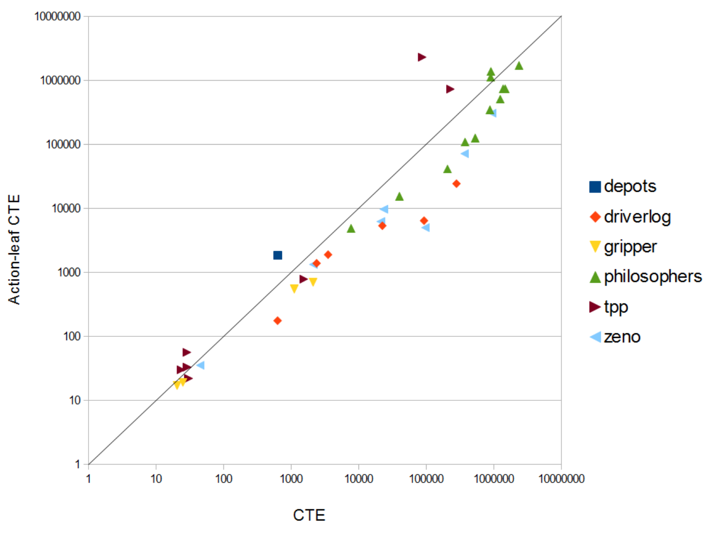


Figure 5.8: Times on problems solved using the CTE against Action-leaf CTE; solving QBFs with QUBE7.0; times in ms.

5.1.5 Comparison with SAT-based encodings

The same problems were solved using two SAT-based solvers: SATPLAN'04 and MADAGASCAR [86].

SATPLAN'04 was chosen as it uses the same state-representation as the CTE and Flat encoding. All three encodings are state-based encodings using an actions-only state representation, as described in Section 2.1.1.

It is possible for alternative state representations to be used with the CTE or Flat encoding, such as the fluent and action encoding of SATPLAN'06 [59]; split action representation of SOLE [92]; or SAS$^+$ representation of SASE [50]. However, in order to obtain the clearest measure of the gap between SAT and QBF-based techniques it is necessary only that the state representations are the same and that both approaches use the same top-level strategy, solver modifications and embedded Planning-specific knowledge. SATPLAN'04 and both QBF approaches use the same state representation and top-level strategy, shown in Figure 2.10, and no other modifications.

The results from MADAGASCAR represent the current state-of-the-art for Planning as SAT. These results are useful as a comparison between SATPLAN'04 and more sophisticated SAT-based Planners. MADAGASCAR includes a modified solver (Figure 2.9), alternative top-level strategy (Figure 2.14), and $\exists$-step parallel-step semantics. A full description of the Planner can be found in Section 2.1.3 and in Rintanen 2010 [86]. These improvements could be included using both CTE and Flat encoding approaches. It is also important to note that MADAGASCAR does not provide optimal solutions.

Comparing Solving Times

The experiments were run under the same conditions and time limits as before. The results for SATPLAN'04 are displayed in Table 5.12 and Figure 5.9. The QDPLL solvers scale much more poorly than SATPLAN'04, as expected. Quantor also scales more poorly, but performs comparably to SAT, with almost identical performance on the easier instances. This is also to be expected, as Quantor resolves the QBF to SAT during the solution process.

These results show a clear gap in the performance between the CTE and SAT. QBF encodings with exponential time-steps are not yet competitive with SAT techniques.

Also presented in Table 5.12 and Figure 5.9 are the times taken to solve using an Action-Leaf state representation: a state representation not possible to apply in a SAT-based encoding. These encodings were solved using QuBE7.0. As can be seen from the table and figure, the alternative state representation is an improvement over that used by SATPLAN'04, when solving the CTE using QuBE7.0.

Figure 5.10 compares the solution times for SATPLAN'04 and MADAGASCAR. As can be seen from the figure, the state-of-the-art solver scales far better on the benchmark problems. The solver, although not finding makespan optimal solutions, finds solutions within five time-steps of makespan optimal up to 100 times faster than SATPLAN'04. The possibility of applying the improvements of MADAGASCAR (modified solver, alternative top-level strategy, and $\exists$-step semantics) to the QBF approaches is discussed in for future work in Section 6.2.

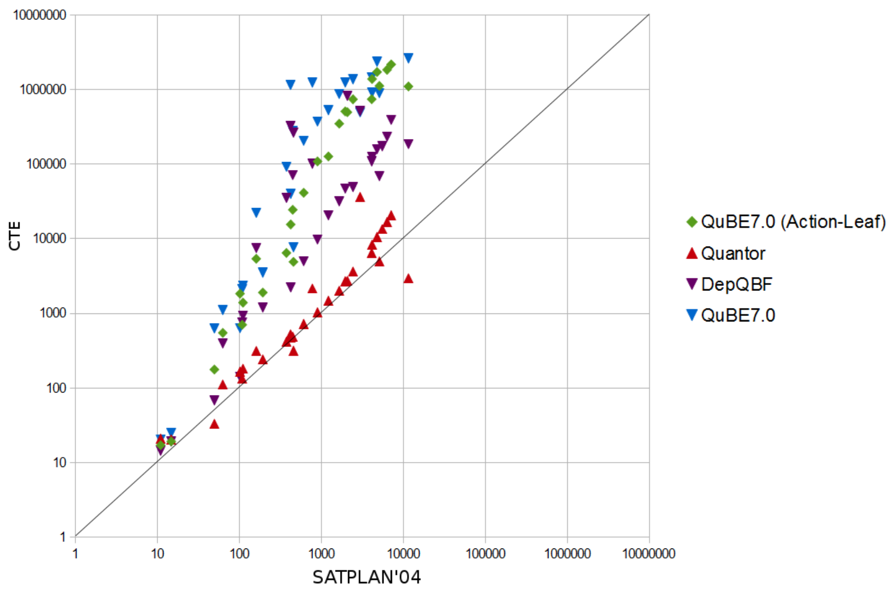


Figure 5.9: Solution times for QBF encodings (CTE) against solution times for SATPLAN'04, solving QBFs with a variety of solvers; times in ms.

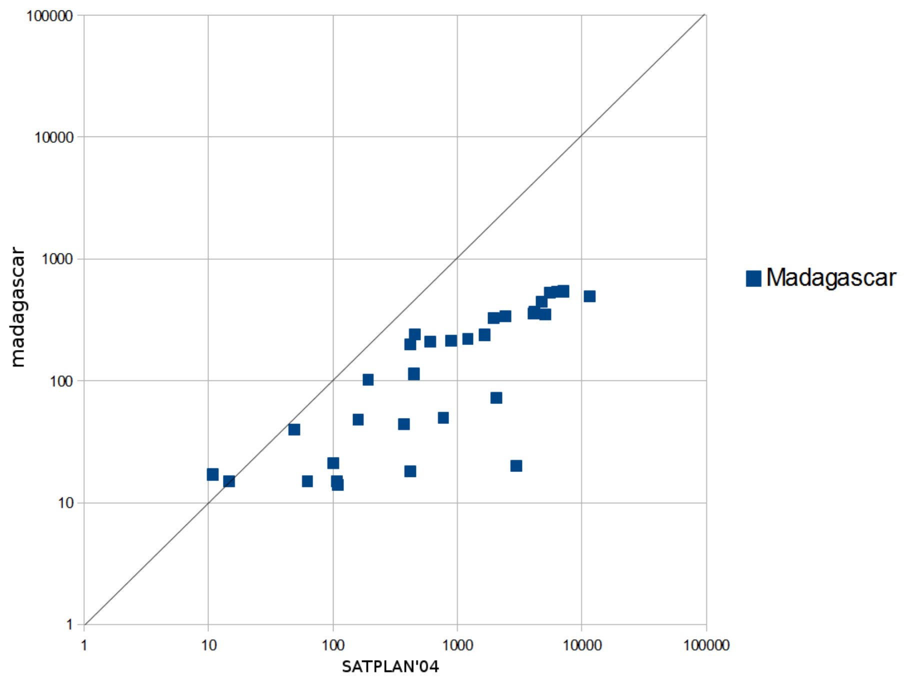


Figure 5.10: Solution times for Madagascar against solution times for SATPLAN'04; times in ms.

Problem	SATPLAN	Quantor	QuBE7.0 Action-leaf	QuBE7.0	DepQBF
depots01	101	166	1839	636	140
depots02	773	2148	-	1239214	101095
driverlog01	49	33	177	628	68
driverlog03	110	181	1396	2373	932
driverlog04	373	419	6477	92483	35319
driverlog05	418	524	-	1152402	323176
driverlog06	192	242	1905	3507	1195
driverlog07	159	314	5404	22302	7499
driverlog08	445	488	24538	279380	71619
driverlog10	2069	2723	498757	-	821889
freecell01	2961	36541	-	496439	513071
gripper01	14	21	19	25	19
gripper02	10	21	17	20	14
gripper03	62	111	549	1108	395
gripper04	107	133	705	2104	762
opticaltelegraph01	11518	2952	1104844	2609927	186104
philosophers01	455	317	4911	7680	265193
philosophers02	421	471	15595	39943.5	2232
philosophers03	606	725	41426	205569	5006
philosophers04	893	1037	109151	374160	9817
philosophers05	1215	1484	127132	528924	20622
philosophers06	1647	2014	349801	873563	31684
philosophers07	1949	2708	510493	1243037	47097
philosophers08	2423	3630	742869	1370321	49323
philosophers09	5092	4988	1121133	896935	69093
philosophers10	4102	6458	744713	1470890	108724
philosophers11	4144	8342	1388551	904039	124423
philosophers12	4765	10606	1724330	2354893	158224
philosophers13	5516	13452	-	-	173314
philosophers14	6317	16821	1846049	-	234561
philosophers15	7084	20714	2178180	-	393328

Table 5.12: Time taken to solve instances using the CTE and SATPLAN'04, solving QBFs with with a variety of solvers; times in ms.

Comparing Memory

The memory usage was recorded for SATPLAN'04 over small time intervals during the solving of each domain. The average value of the readings on mutually solved problems was was taken for each domain – this differs from the previous memory results for the QBF-based encodings, which were taken per instance; the decreased time to solve for SATPLAN'04 proved too short to take sufficient readings per problem. The QBF encodings were solved using DepQBF.

The average memory used is charted in Figure 5.11. It can be seen that the QBF approach has a much smaller memory footprint than SAT, using roughly a fifth of the memory used by the SAT-based solver. This is to be expected, as by using QBF to encode Planning problems we are trading time for space.

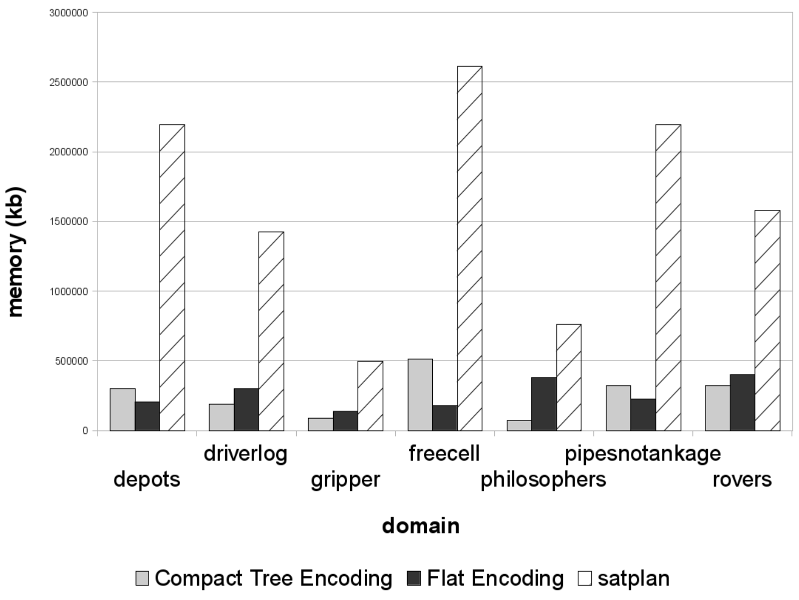


Figure 5.11: Average memory usage over time for SAT and QBF encodings per domain, solving QBFs with DepQBF, SAT instances with PICOSAT; memory in kb.

5.2 Solving Partially Grounded QBF Encodings

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

- as the size of the problem increased, the Partially Grounded QBF (PGQBF) approach would scale better than the SAT approach in both encoding time and solving time;
- as the size of the problem increased, PGQBF 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 PGQBF.

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.

For each domain a number of problems were generated, gradually increasing the size of the domain. These problems were then translated into SAT and PGQBF encodings. The time-step bound for each encoding was the smallest at which the formula is satisfiable. The time taken for this translation was recorded. We used the SAT encoding used by SATPLAN'06 [59] as it used the same STRIPS-based fluent/action representation as the PGQBF 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 5.13. This table highlights the difference in scaling between the approaches. The size of the SAT encoding very quickly becomes unreasonable, while the PGQBF encodings scale much better; in the case of the pigeonhole problem PGQBF encoding grows linearly with the number of objects. This linear growth is due to the increased number of time-steps in the optimal solution.

Table 5.13 shows that there are a number of problems that cannot be encoded in SAT using the 2 hour time limit given, but can be encoded as PGQBF.

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

The encodings were solved using the SAT solver *picosat-535* [5] and the QBF solver *quantor-3.0* [4]. The times are recorded in Table 5.14 for both encoding and solving. The experiments were all given a time limit of 2 hours and run on a machine with 8GB of RAM (no artificial bound on the amount of memory was used).

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

As can be observed, PGQBF 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 or pigeonhole64 in the time available. PGQBF 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 times, although the PGQBF curve increases more slowly than the SAT curve. Neither approach was able to solve gripper32. These results support our first and second hypotheses, that PGQBF scales better than SAT in both encoding and solution time, and that PGQBF 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 PGQBF, within a fixed time limit.

Chapter 6

Conclusion

As stated in the Introduction, this thesis aims to explore the idea of encoding classical Planning problems as Quantified Boolean Formulae. Two different approaches were presented, in Chapter 3 and Chapter 4. These approaches were evaluated in Chapter 5.

This section is split into three parts: first summarizing the contributions of this thesis; then presenting ideas for future work relating directly to the encodings, that is, improving upon or combining the approaches; and finally detailing future work that is motivated by, but not an extension of, the encodings themselves.

6.1 Summary

6.1.1 The Compact Tree Encoding

The Compact Tree Encoding (CTE) is a novel approach to translating Classical Planning into Quantified Boolean Formulae.

Similar to Rintanen's translation that we call the Flat encoding, it encodes Planning problems in a Boolean formula with a fixed makespan that is exponential in the number of states represented. This property is the key attraction for both the Flat encoding and the CTE. Our encoding is novel in its construction, in that the plan trajectory corresponds to a traversal of the tree prescribed by the expansion of the universal quantifiers.

Implicit in the design of the encoding is the idea that constraints – in this case transition constraints – can be made between two nodes of the QBF tree that are formed from the same variables. This is done by using an intermediary set of variables higher in the quantification order – in the CTE these variables represent the midpoint state between the two nodes.

SAT-based planners work on encodings in which every variable represents some aspect of the Planning problem. However, in the Flat encoding the majority of the variables are redundant – essentially serving only as machinery to allow the problem to be represented in QBF. In the CTE we include only variables that represent distinct aspects of the underlying Planning problem, much like SAT.

As a result the CTE uses fewer universally and existentially quantified variables than the Flat encoding when encoding problems with a given makespan.

In fact, for plans encoded with the same depth of universal quantification, the Compact Tree Encoding describes a plan of double the makespan of that described by the Flat encoding.

We consider this to be the more natural translation of Planning to QBF, as well as the more elegant.

We have experimented with sets of problem instances taken from the IPC benchmarks, and we have shown that the CTE leads to both a faster solution (and proof of unsolvability) of the harder problems in these sets and on average uses less overall memory in doing so. Our experiments also show that the improved time to find a solution during iterative deepening stems from the makespans at which the CTE requires one less quantifier alternation – a product of its unique structure.

6.1.2 Leaf-Based Encodings

The Leaf-Based Encodings are extensions of the CTE that use novel state representations specific to QBF encodings logarithmic in the number of time-steps.

These state representations, the Action-leaf and Fluent-leaf representations, exploit the tree structure of the CTE in order to further reduce the number of variables and clauses required to model a state-based reachability problem.

6.1.3 The Partially Grounded QBF Encoding

The PGQBF encoding is a novel approach to translating Classical Planning into QBF.

The PGQBF encoding actually introduces a general approach to lifting SAT encodings of Planning problems – and other problems – into QBF. The encoding presented in Chapter 4 is an example of this general approach applied to the SAT-based translations of Kautz et al. [55, 56]. The same approach is applicable to any of the state-based representations discussed in Section 2.1.1, or either of the QBF encodings presented in Chapter 3: the CTE and the Flat encoding.

Lifting objects into equivalence classes is a very powerful idea, being explored in different ways by a number of researchers in Planning [72, 78]. It is well-known that grounding of 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.

The PGQBF approach is inspired by earlier work in least-commitment Planning. The approach is used 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 makes use of a number of novel constraints in order to ensure consistent reasoning. Those relating to the operator lock variables are the most important. These variables allow the formula to include a disjunction between parts the model that are represented by the same variables. In the case of the PGQBF this means imposing a disjunction on a set of actions that are represented using the same variable. Without the lock variables, and the constraints equating them to the universal variables, this type of constraint is not possible. It should be emphasised that this approach, integral to the PGQBF encoding, is applicable in modelling as QBF in many other scenarios.

In Chapters 3 and 4 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. In fact, our hypotheses that

- as the size of the problem increased, the PGQBF approach would scale better than the SAT approach in both encoding time and solving time;
- as the size of the problem increased, PGQBF would find solutions faster than the SAT approach; and
- 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 PGQBF.

were completely validated.

6.2 Future Work

6.2.1 Extending the Encodings

In this section we describe possibilities for improvements to the encodings and Planning approaches described in this thesis, along with some improvements that have already been implemented, but not yet evaluated.

Alternative Top-level strategies for the CTE

Ray and Ginsberg [76] introdued the idea of formula re-use, as described in Section 2.1.3. Their approach has been applied directly to the CTE with only minor adjustments.

In CRICKET the SAT formula contains goal constraints at multiple levels and a modified SAT solver attempts to satisfy these in an order that ensures the plan is of an optimal makespan. If no plan is found, the makespan bound is increased by some value, larger than one. Similarly, a QBF solver – QUBE7.0 – has been modified to branch upon specified literals in the outermost layer. We call this branching order QBF-preferences. For example, in the following QBF in QDIMACS format

```
c f 1 −2
e 1 2
−1 2
```

the first line contains the QBF-preferences; an ordered list of literals that define the initial branching order of the (QDPLL) solver. Ordinarily there would be three satisfying assignments: $(-1, 2)$; $(-1, -2)$; and $(1, 2)$. However, the variable 1 is branched upon first, leading to only the third solution – as long as the solver reasons with QBF-preferences.

With the use of these preferences the CTE can be modified to return a makespan optimal solution within the maximum bound described by the formula. If no solution is found then the number of quantifier alternations is incremented and the process repeated.

This technique is attractive for the QBF approach as it conforms to the existent structure of the CTE; the makespan bound of the CTE doubles with each iterative quantifier alternation. Without this technique a CTE with k

alternations can represent a plan with up to $2^{k+1} - 1$ states. In order to search for plans with makespan n, where $2^k - 1 < n < 2^{k+1} - 1$, k alternations must be used. Applying the layered formula technique with QBF-preferences diminishes this redundancy.

The technique has been implemented. We use QuBE-p, our modified version of QuBE7, to search with QBF-preferences. The encoding with preferences, which we call a layered encoding, is the CTE with a state representation consisting of both fluent and action variables, as used by SATPLAN'04.

Structural Improvements to the CTE

Windowing is the name given to extending the nodes of the CTE to describe additional states. A CTE with windowing in the innermost quantification level (the leaf nodes) would have the quantification layer:

$$\exists X_k \forall y_k \ldots \exists X_1 \forall y_1 \exists X_{0,0}, X_{0,1}, X_{0,2}.$$

In this case the size of the windows is three. Transition constraints are added between the windowed states:

$$\tau(X_{0,0}, X_{0,1}) \wedge \tau(X_{0,1}, X_{0,2}).$$

Windowing is useful as it:

1. allows us to more finely control the number of states represented by the formula;
2. allows us to increase the number of states represented without increasing the number of quantifier alternations; and
3. provides opportunity to exploit domain-specific knowledge within the windows, or in the placement of the windows.

(1) and (2) are obvious; in the above example the number of represented states is increased from $2^{k+1} - 1$ without windowing, to $2^{k+2} - 1$. Placing the windows of size one or two in non-leaf states allows for any number of states in between to be represented.

Of particular interest is the idea of using windowing to manage iterative deepening, rather than constraints or a layered encoding, as described above.

Exploiting domain-specific knowledge through the use of windows depends upon the other techniques applied. As an example the londex constraints [16, 17] described in Section 2.1.1 are difficult to apply to the CTE directly. In particular we stated that:

> the encodings in Chapter 3 do not explicitly represent every time-step with unique variables. Due to this it is not possible to simply represent a constraint between elements in separate time-steps without the inclusion of more variables in an earlier quantifier set.

However, with windowing it becomes possible to include constraints that occur between states at set distances, i.e. the size of a window.

The technique has been implemented, although with no consideration to domain-specific knowledge, and has not yet been evaluated.

New representations with the PGQBF

The encoding presented in Chapter 4 uses a state-based representation described by Kautz et al. [55, 56] augmented with simply-split operators and propositions, in a semi-parallel setting. This is far from the state-of-the-art in state representation, as discussed in Section 2.1.1.

The PGQBF approach can be used with a number of state representations. The most promising, with respect to the PGQBF approach are:

- the SAS^+-based encoding of Huang et al. [50, 51]; and
- the split representation of Robinson et al. [92].

Both representations are ideal for partial grounding. In particular the SAS^+-based encoding is already formulated in terms of domain-transition and causal graphs – in line with related work in domain level lifting [78].

Automatic generation of PGQBF encodings

Currently PGQBF encodings are hand coded, as the choice of objects to remain ungrounded is carried out manually. In order for the approach to be integrated into a fully automated Planning system the process of deciding which objects to lift into equivalence classes, if any, must be carried out automatically. This decision can be made easily, as the number of variables and clauses required for grounding can be calculated in advance and weighed against the overhead for lifting the object.

6.2.2 Related Work

In this section we describe works that are motivated by, or are other applications of, the encodings and Planning approaches presented in this thesis.

QDPLL solvers and QBF encodings with exponential time-steps

As described in Section 2.3.1 QDPLL solvers perform poorly on the CTE and Flat encoding. This is due to the top-down nature of the search, and the limited propagation that results. This can be illustrated in a small example; consider the following QBF:

$$\exists x_1 \forall x_2 \exists x_3 ((\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)).$$

This QBF has only two solutions: $\{\neg x_1, \neg x_3[\neg x_2], x_3[x_2]\}$ and $\{\neg x_1, \neg x_3\}$. That the variable x_1 must satisfy the first clause is clear, as we know that under some contexts x_3 must be false. This becomes obvious when the universal variables are expanded:

$$\begin{array}{l} \exists x_1, x_3(\\ \quad (\neg x_1 \vee x_3) \wedge \\ \quad (\bot \vee \neg x_3) \wedge \\ \quad (\top \vee \neg x_3)). \end{array}$$

However, this information is not propagated and QuBE7.0 [41] will attempt to first assign $x_1 \models \top$, leading to a conflict, and backtrack. This sort of combination of constraints – in which an inner existential variable is constrained only under some contexts and influences the values of outer existential variables – appears in the CTE as the initial state and goal constraints. That this information is not propagated until the innermost layers are reached is a huge disadvantage for this combination of encoding and solver.

Improvements to QDPLL solvers – specifically propagation methods motivated by this example – while not directly related to the work presented in this thesis, have the potential to greatly improve the performance of these approaches on Planning problems.

Model-Checking

It has already been noted that the approaches, although discussed in the context of Planning, are applicable to any state-based reachability problem. In fact the QBF encoding presented by Papadimitriou [74] and implemented by Dershowitz et al. [22], was used as a proof of the PSPACE-complexity of the Reachability problem. Dershowitz et al. made encodings of instances of Model-Checking problems; applying the encodings presented in this thesis to the field Model-Checking has been suggested, and left for future work.

6.3 Final Word

The abstract of this thesis states that QBF is a promising setting for Planning given that the problems have the same complexity. This similarity enables us to generate encodings that, while less readable by human standards, describe the state space and plan trajectory in a manner as succinct as the problem description itself.

The CTE, although not yet competitive with SAT on the benchmark problems, furthers an interesting avenue of research: both forming the foundation for more efficient and competitive encodings, such as the Leaf-based Encodings and QBFs including top-level strategies such as used by Ray and Ginsberg [76]; and providing motivating examples for QBF solving strategies – most importantly highlighting the inability of top-down QDPLL solvers to propagate any information from clauses involving only a single existential variable, unless it is in the outermost quantification level.

PGQBF encodings are an orthogonal direction of research and have already proved competitive with SAT-based approaches on the domain in which object-level lifting is expected to provide the greatest advantage.

As QBF solving matures these encodings are likely to become ever more viable alternatives to the SAT-based approaches. Improvements to the encodings can be advised by the popular methods of solution, while at the same time the encodings themselves provide difficult models with which to challenge the solvers.

The aim of this thesis is to explore the idea of encoding classical Planning problems as Quantified Boolean Formulae. This has been achieved; the thesis explores in a number of ideas, specific to Planning in their description, but applicable to a large number of problems and suitable for most improvements and strategies employed by SAT-based approaches.

6.4 List of Publications

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

2. M. Cashmore, M. Fox, and E. Giunchiglia. Planning as quantified boolean formulae. In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'11)*, 2011

3. M. Cashmore and M. Fox. Partially grounded planning as quantified boolean formula. In *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS'12)*, 2012

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

5. M. Cashmore, M. Fox, and E. Giunchiglia. Partially grounded planning as quantified boolean formulae. *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 2013

Bibliography

[1] F. Bacchus. Overview of the AIPS-00 planning competition, 2000. http://www.cs.toronto.edu/aips2000/, last accessed Jan. 2013.

[2] C. Bäckström. Equivalence and tractability results for SAS+ planning. In *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning (KR'92)*, pages 126–137, 1992.

[3] E. Best and R. Devillers. Sequential and concurrent behaviour in petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.

[4] A. Biere. Resolve and expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT04)*, pages 238–246, 2004.

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

[6] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Design Automation Conference (DAC'99)*, pages 317–320, 1999.

[7] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. 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)*, pages 193–207, 1999.

[8] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1636–1642, 1995.

[9] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[10] M. Cadoli, M. Schaerf, M. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

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

[12] M. Cashmore and M. Fox. Partially grounded planning as quantified boolean formula. In *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS'12)*, 2012.

[13] M. Cashmore, M. Fox, and E. Giunchiglia. Planning as quantified boolean formulae. In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'11)*, 2011.

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

[15] M. Cashmore, M. Fox, and E. Giunchiglia. Partially grounded planning as quantified boolean formulae. *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 2013.

[16] Y. Chen, R. Huang, Z. Xing, and W. Zhang. Long-distance mutual exclusion for planning. *Artificial Intelligence*, 173(2):365–391, 2009.

[17] Y. Chen, Z. Xing, and W. Zhang. Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1840–1845, 2007.

[18] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *25 Years of Model Checking*, pages 196–215, 2008.

[19] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pages 21–27, 1993.

[20] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[21] M. De Luca, E. Giunchiglia, M. Narizzano, and A. Tacchella. "Safe planning" as a QBF evaluation problem. In *Proceedings of the 2nd Robocare Workshop*, ISTC-CNR, 2005.

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

[23] V. Diekert and Y. Mtivier. Partial commutation and traces. *Handbook of Formal Languages*, 3:457–534, 1997.

[24] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Recent Advances in AI Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 169–181. 1997.

[25] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

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

[27] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1-2):75–88, 1995.

[28] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, 2000.

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

[30] J. Finger. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982.

[31] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, pages 83–91, 2002.

[32] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelliigence Research*, 9:367–421, 1998.

[33] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 956–961, 1999.

[34] M. Fox and D. Long. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[35] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming: The 5th International Conference and Symposium*, pages 1070–1080, 1988.

[36] A. Gerevini. The 5th international planning competition. *ICAPS'06 Report*, 1999.

[37] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 948–953, 1998.

[38] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas Satisfiability Library (QBFLIB), 2001. www.qbflib.org, last accessed Jan. 2013.

[39] E. Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, pages 657–666, 2000.

[40] E. Giunchiglia, P. Marin, and M. Narizzano. An effective preprocessor for QBF pre-reasoning. In *Proceedings of the 2nd International Workshop on Quantification in Constraint Programming (QiCP'08)*, 2008.

[41] E. Giunchiglia, P. Marin, and M. Narizzano. QuBE7.0 system description. *Journal of Satisfiability.*, 7(8):83–88, 2010.

[42] F. Giunchiglia and P. Traverso. Planning as model checking. In *Proceedings of the 5th European Conference on Planning (ECP'99)*, pages 1–20, 1999.

[43] R. P. Goldman, M. S. Boddy, and L. Pryor. Planning with observations and knowledge. In *Theories of Action, Planning, and Robot Control: Bridging the Gap: Papers from the 1996 AAAI Workshop*, pages 78–85, 1996.

[44] A. Goultiaeva, V. Iverson, and F. Bacchus. Beyond CNF: A circuit-based QBF solver. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, pages 412–426, 2009.

[45] K. Heljanko. Bounded reachability checking with process semantics. In *Proceedings of the 12th International Conference on Concurrency Theory (Concur'01)*, pages 218–232, 2001.

[46] M. Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 161–170, 2004.

[47] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[48] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

[49] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22(1):215–278, 2004.

[50] R. Huang, Y. Chen, and W. Zhang. A novel transition based encoding scheme for planning as satisfiability. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

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

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

[53] S. Kambhampati. Challenges in bridging plan synthesis paradigms. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 44–49, 1997.

[54] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the 5th International Conference of Principles on Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

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

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

[57] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS'98*, 1998.

[58] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, 1999.

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

[60] F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver system description. *Journal on Satisfiability, Boolean Modeling and Computation.*, 7:7176, 2010.

[61] F. Lonsing and A. Biere. Integrating dependency schemes in search-based QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT'10)*, pages 158–171, 2010.

[62] D. Magazzeni. *Explicit Model Checking Techniques applied to Control and Planning Problems.* PhD thesis, University of L'Aquila, 2009.

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

[64] P. Marin, E. Giunchiglia, and M. Narizzano. Conflict and solution driven constraint learning in QBF. In *Doctoral Program of Constraint Programming Conference*, 2010.

[65] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[66] D. Mcallester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'91)*, volume 2, pages 634–639, 1991.

[67] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[68] D. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 1998.

[69] K. L. McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, 1992.

[70] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[71] H. Nabeshima, T. Soh, K. Inoue, and K. Iwanuma. Lemma reusing for SAT based planning and scheduling. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pages 103–113, 2006.

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

[73] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming (JICSLP'96)*, pages 289–303, 1996.

[74] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[75] J. Porteous and L. Sebastia. Extracting and ordering landmarks for planning. In *Proceedings of UK Planning and Scheduling SIG Workshop (Plansig'00)*, 2000.

[76] K. Ray and M. L. Ginsberg. The complexity of optimal planning and a more efficient method for finding solutions. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 280–287, 2008.

[77] S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI'08)*, pages 975–982, 2008.

[78] B. Ridder and M. Fox. 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)*, 2011.

[79] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[80] J. Rintanen. 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)*, pages 362–376, 2001.

[81] J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 32–41, 2003.

[82] J. Rintanen. Evaluation strategies for planning as satisfiability. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 682–687, 2004.

[83] J. Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI'07)*, pages 1045–1050, 2007.

[84] J. Rintanen. Heuristic planning with SAT: Beyond uninformed depth-first search. In *Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence (AI'2010)*, pages 415–424, 2010.

[85] J. Rintanen. Heuristics for planning with SAT. In *Proceedings of the 16th international conference on Principles and Practice of Constraint Programming (CP'10)*, pages 414–428, 2010.

[86] J. Rintanen. Madagascar: Efficient planning with SAT. In *The 7th International Planning Competition: Description of Participant Planners of the Deterministic Track*, 2010.

[87] J. Rintanen. Heuristics for planning with SAT and expressive action definitions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 2011.

[88] J. Rintanen. Planning with specialized SAT solvers. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI'11)*, 2011.

[89] J. Rintanen, K. Heljanko, and I. Niemel. Parallel encodings of classical planning as satisfiability. In *Logics in Artificial Intelligence*, volume 3229 of *Lecture Notes in Computer Science*, pages 307–319. 2004.

[90] J. Rintanen, K. Heljanko, and I. Niemel. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170:1031–1080, 2006.

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

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

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

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

[95] M. J. Streeter and S. F. Smith. Using decision procedures efficiently for optimization. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 312–319, 2007.

[96] M. Wehrle and J. Rintanen. Planning as satisfiability with relaxed ∃-step plans. In *Proceedings of the 20th Australian joint conference on Advances in Artificial Intelligence (AI'07)*, pages 244–253, 2007.