On Conflicts and Strategies in QBF

Nikolaj Bjørner[1], Mikoláš Janota[2] and William Klieber[3]

[1] Microsoft Research, Redmond, USA
[2] Microsoft Research, Cambridge, UK
[3] CERT/SEI, Carnegie Mellon University

Abstract

QBF solving techniques can be divided into the DPLL-based and expansion-based. In this paper we look at how these two techniques can be combined while using strategy extraction as a means of interaction between the two. Once propagation derives a conflict for one of the players, we analyse the proof of such and devise a strategy for the opponent. Subsequently, this strategy is used to constrain the losing player. The implemented prototype shows feasibility of the approach. A number of avenues for future research can be envisioned. Can strategies be employed in a different fashion? Can better strategy be constructed? Do the presented techniques generalize beyond QBF?

1 Introduction

Qualified Boolean formulas (QBFs) can be seen from a number of different perspectives: as two-player games, as a canonic PSPACE problem, or as a means of studying quantification in general. This enables QBF to model a number of interesting problems and motivates the development of efficient QBF solvers [4, 23]. Existing QBF solvers reveal an interesting dichotomy. Some solvers *traverse the search* space by assigning values to the individual variables similarly as a backtracking algorithm would [30, 11, 21]. Other solvers *expand quantifiers* into logical connectives according to their semantics [20, 6, 14]. In this paper we aim to bridge this dichotomy by guiding quantifier expansion by conflicts found by traversal. The approach is motivated by another well known phenomenon. If we look at a QBF as a two-player game, solving a formula equates to finding a winning strategy of one of the players. Interestingly enough, it is easy to construct a formula with a very simple winning strategy for one of the players but the formula itself is very hard to solve for current solvers. This suggests that we look for winning strategies *explicitly* rather than relying on them being discovered by search. The strategy extraction we propose is anchored in *conflict analysis.* Once it is discovered that one of the players loses in a certain part of the search space, the proof for such is used to devise a strategy for the opposing player. Subsequently, this strategy is used to partially expand the formula.

2 Preliminaries

Elementary notion of logic is assumed. Here we review the notation and concepts used throughout the paper.

2.1 Propositional Logic

Propositional variables are typically denoted by letters from the end of the alphabet, e.g. x, y_1. A *literal* is a variable or its negation, e.g. $x, \bar{z}$, with $\bar{\bar{z}} = z$ A *clause* is a disjunction of finitely many literals. The empty clause is denoted by $\perp$. A formula in *conjunctive normal form (CNF)* is a conjunction of clauses. Whenever convenient, a CNF formula is treated as a set of clauses, e.g. $\phi = \{(x \vee z), (\bar{z})\}$. By a *substitution* we mean a mapping τ from propositional variables to Boolean expressions. For a formula ϕ and a substitution τ we write $\phi|_\tau$ for the application of τ to ϕ, i.e. a simultaneous replacement of each $x \in \mathtt{dom}(\tau)$ by $\tau(x)$. Substitutions are denoted as $0/u, x/u$ etc.

2.2 QBF

Quantified Boolean formulas extend propositional logic by the universal and existential quantifiers ($\forall$,$\exists$), with the standard semantics $\exists x.\, \phi = \phi|_{0/x} \vee \phi|_{1/x}$ and $\forall x.\, \phi = \phi|_{0/x} \wedge \phi|_{1/x}$. Note that $QxQy.\, \phi = QyQx.\, \phi$ for $Q \in \{\exists, \forall\}$. A formula is *closed* if no variable appears unquantified.

A QBF is in *prenex form* if it is in the form $\Phi = Q_1 X_1, \ldots, Q_n X_n.\, \phi$ where $Q_i \in \{\forall, \exists\}$ and X_i are sets of variables and ϕ is purely propositional. The formula ϕ is referred to as the *matrix* of the formula. Unless specified otherwise, prenex form closed formulas are assumed. For a variable x, we write $\mathsf{qlevel}(x)$ for the quantification level of x, i.e. $\mathsf{qlevel}(x) = i$ iff $x \in X_i$. Analogously, for a literal ℓ, we write $\mathsf{qlevel}(\ell)$ for $\mathsf{qlevel}(x)$ if $\ell = x$ of $\ell = \bar{x}$. For a clause C and $Q \in \{\exists, \forall\}$ we write $\mathsf{qlevel}_Q(C)$ for the maximal quantification level appearing in C that belongs to Q, i.e. $\mathsf{qlevel}_Q(C) = \max(\{\mathsf{qlevel}(\ell \in C) \mid \ell \text{ belongs to } Q\})$. If C contains no Q literals, $\mathsf{qlevel}_Q(C) = \bot$.

It is often convenient to view a QBF as a game between the existential and the universal player. The existential player assigns values to the variables that are existential he qualified and the universal player assigns values to the universally quantified variables. The players assign values to variables in the order of the prefix, starting from the outermost to the innermost. A play is won by the existential player if the matrix evaluates to true and is won by the universal player if the matrix evaluates to false. A formula is true if and only if there exists a winning strategy for the existential player. The concept of strategies is formalised by the following definition.

Definition 1 (strategy and winning strategy)**.** *A Boolean function* S_x *is* strategy for a variable x *if the input of* S_x *are the variables* $\{y \mid \mathsf{qlevel}(y) < \mathsf{qlevel}(x)\}$*. A* strategy $\mathscr{S}$ for player $Q \in \{\exists, \forall\}$ *is a set of strategies* S_x *for variables* x *belonging to the player* Q*. For a formula* $\Phi = Q_1 X_1, \ldots, Q_n X_n.\, \phi$*, a strategy* $\mathscr{S}$ *for* $\exists$ *is a* winning strategy *if* $\bigwedge_{\mathrm{S}_x \in \mathscr{S}} (x = \mathrm{S}_x) \models \phi$*. A strategy* $\mathscr{S}$ *for* $\forall$ *is a* winning strategy *if* $\bigwedge_{\mathrm{S}_x \in \mathscr{S}} (x = \mathrm{S}_x) \models \neg\phi$*.*

Theorem 1. *A closed QBF in prenex form is true if and only if there exists a winning strategy* $\mathscr{S}$ *for the existential player.*

Corollary 1. *For any closed QBF in prenex form* Φ *there exists a winning strategy for one and only of the players. The existential player has a winning strategy for* Φ *iff it does not have a winning strategy for* $\neg\Phi$*.*

2.3 Propagation and Learning in QBF

We next recall propagation and conflict learning in the context of QBF. We will model the propagator by the following three functions. $\mathtt{Propagate}(\delta, \Phi)$ infers new facts from formula Φ, based on the decisions δ. $\mathtt{StrengthenPropagator}(\Phi, Q, C)$ makes sure that player Q satisfies the clause C in the future. Finally, the function $\mathtt{Learn}(\mathsf{loser}, \delta, \Phi)$ learns a new clause for a player loser in the case when the propagator reaches a conflict for loser.

These operations can be seen as an extension of traditional CDCL SAT solving [25, 24] to QBF [30, 12, 10] and are discussed in more detail in turn.

2.3.1 Propagate

Given the current set of decisions, a propagator derives some necessary facts for either of the players. If this set of decisions leads to a *conflict*, the propagator learns a clause that prevents this conflict in the future. By a conflict, we mean that it is deemed that one of the players can no longer win the play. Recall that if the current set of decisions satisfies the matrix, the universal player loses. Hence, some authors talk about *solutions* and *conflicts* to distinguish the two losing scenarios. In this paper we always talk about conflicts for both of the players.

The function $\texttt{Propagate}(\delta, \Phi)$ is used to model propagation, given the decisions δ on a QBF Φ. The function returns a pair (τ, loser), where τ is the set of derived facts. If the propagation leads to a conflict, $\tau = \bot$ and $\mathsf{loser} \in \{\exists, \forall\}$ contains the player for which the conflict occurred. For instance, $\texttt{Propagate}(\{e_1\}, \exists e_1 e_2 \forall u \exists e_3 e_4.\, (\bar{e}_1 \vee e_2) \wedge (u \vee e_3 \vee e_4))$ would return $(\{e_1, e_2, \bar{u}\}, -)$.

Propagation can be realized in a number of different ways [30, 31, 19, 21, 12, 22]. For this presentation we mainly rely on the approach of Goultiaeva et al. [12], with some ingredients introduced by Klieber [19, 18]. We note, however that the presented algorithm is not limited to this particular implementation of propagation. The input is a QBF $(\forall Y_1 \exists X_1 \ldots \forall Y_n \exists X_n.\, \phi)$, not necessarily with a CNF matrix. The propagator produces two *sub-propagators*—one for each player. One by translating ϕ to CNF and the second by translating $\neg\phi$ to CNF while flipping the quantifiers. Formulas are translated to CNF by introducing fresh (Tseitin) variables [28]. Hence, obtaining $(\forall Y_1 \exists X_1 \ldots \forall Y_n \exists X_n T_1.\, \psi)$ and $(\exists Y_1 \forall X_1 \ldots \exists Y_n \forall X_n \exists T_2.\, \xi)$ such that $(\exists T_1.\, \psi) = \phi$ and $(\exists T_2.\, \xi) = \neg\phi$. This enables symmetrical CNF-based propagation for both players.

$\texttt{Propagate}(\delta, \Phi)$ is implemented by pushing δ onto both sub-propagators. Once a sub-propagator infers a new fact, this fact is pushed onto the other one. New facts are derived until saturation. Tseitin variables are treated somewhat specially. Each sub-propagator may infer facts regarding Tseitin variables but these are not pushed onto the other sub-propagator nor reported on the output of $\texttt{Propagate}$.

In this setting, both sub-propagators work from the perspective of the existential player. In another words, both sub-propagators try to satisfy their respective formulas. A sub-propagator sets an existential literal $\ell \in C$ to true, if the clause C satisfies all the following conditions:

1. There is no literal set to true in C.
2. All existential literals of C except for ℓ are false, i.e. ℓ is unassigned.
3. All universal literals at quantification level $< \mathsf{qlevel}(\ell)$ are false.

A conflict is reached when there is a clause whose all existential literals are set to false.

Example 1. *Consider* $\Phi = \exists e_1 \forall u \exists e_2.\, u \vee ((e_1 \vee e_2) \wedge (\bar{e}_2) \wedge (\bar{e}_1 \vee e_2))$. *For the existential sub-propagator we get the formula* $\exists e_1 \forall u \exists e_2.\, (e_1 \vee u \vee e_2) \wedge (u \vee \bar{e}_2) \wedge (\bar{e}_1 \vee u \vee e_2)$. *For the universal sub-propagator a formula of the form* $\forall e_1 \exists u \forall e_2 \exists T.\, (\bar{u}) \wedge \psi$*, with some Tseitin variables* T *and clauses* ψ *encoding the rest of the formula. The universal sub-propagator immediately propagates the unit clause* $\bar{u}$*, which yields* $\bar{e}_2$ *in the existential sub-propagator through the second clause,* e_1 *due to the first clause and finally the third clause gives a conflict. Such conflict is irresolvable for the existential player since it doesn't depend on any of its decision; in fact it didn't make any.*

2.3.2 Strengthening Propagator

The function $\texttt{StrengthenPropagator}(\Phi, Q, C)$ returns a modified Φ such that the player Q is forced to satisfy the clause C. Semantically, this corresponds to strengthening the matrix ϕ as

Algorithm 1: Combined QBF Solving

```
input  : $\Phi = Q_1 X_1 \ldots Q_i X_i \ldots Q_n X_n.\, \phi$
output: truth value of $\Phi$
1  $\delta \leftarrow \emptyset$                                   // initialization of decisions
2  $\alpha \leftarrow [\alpha_1 = \mathsf{true}, \ldots, \alpha_n = \mathsf{true}]$   // initialization of abstraction vector $\alpha$
3  while true do
4      $(\tau, \mathsf{loser}) \leftarrow \mathtt{Propagate}(\delta, \Phi)$         // inferring $\tau$ by QBF propagation
5      if $\tau = \bot$ then // conflict resolution
6          $(k, \alpha, \Phi) \leftarrow \mathtt{LearnAndRefine}(\delta, \Phi, \alpha, \mathsf{loser})$   // refine at level $k$
7          if $k = \bot$ then return $(\mathsf{loser} = \forall)\,?\,\mathsf{true} : \mathsf{false}$   // no recovery
8          $\delta \leftarrow \{l \in \delta \mid \mathsf{qlevel}(l) < k\}$            // backtrack decisions
9      else // decision-making
10         $k \leftarrow$ minimal quantification level not fully assigned in $\tau$
11         $\tau_k \leftarrow \{\ell \in \tau \mid \mathsf{qlevel}(\ell) \leq k\}$      // filtering out irrelevant values
12         $(\mu, \tau') \leftarrow \mathtt{SAT}(\alpha_k \wedge \tau_k)$               // consult $\alpha_k$ for a decision
13         if $\mu = \bot$ then // abstraction unsatisfiable
14             $\Phi \leftarrow \mathtt{ResolveUnsat}(\tau', Q_k, \Phi)$     // update $\Phi$ based on the core $\tau'$
15             if $\Phi = \bot$ then return $(Q_k = \forall)\,?\,\mathsf{true} : \mathsf{false}$   // no recovery
16         else
17             $v \leftarrow$ a variable unassigned by $\tau$ at quantification level $k$
18             $\delta \leftarrow \delta \cup \{\mu(v)\,?\,v : \bar{v}\}$      // make a decision on $v$ based on model $\mu$
```

$\phi \wedge C$, if $Q = \exists$, and weakening it as $\phi \vee \neg C$, if $Q = \forall$. Since we are following the approach of two sub-propagators, this operation in practice corresponds to adding the clause C to the sub-propagator of player Q.

2.3.3 Learn

Given a conflict propagation, the function $\mathtt{Learn}(\mathsf{loser}, \delta, \Phi)$ produces a clause C rectifying the conflict. If the result of Learn is $\bot$, it means that the conflict is irresolvable and the formula is solved (there is a winning strategy for the loser's opponent). For instance,

$$\mathtt{Learn}(\exists, \{e_1\}, \exists e_1 \forall u \exists e_2.\, (\bar{e}_1 \vee \bar{e}_2) \wedge (\bar{e}_1 \vee u \vee e_2))$$

might return the clause $(\bar{e}_1 \vee u)$. In general we assume that the learned clause is such that strengthening the sub-propagator of loser by C does not change the set of winning strategies of loser. In practice, other properties are guaranteed—similar to *unique implication point (UIP)* in SAT (see [30, Sec. 3.3]).

3 Combining Propagation and Refinement

This section presents an algorithm that combines ideas from several existing approaches to QBF. It uses DPLL-style clause learning [30, 12], abstraction & refinement learning of RAReQS [15, 14], and the flat architecture of qesto [16].

In the spirit of standard backtracking QBF algorithm, variables are given values from the outermost ones to the innermost ones. The algorithm maintains a vector of propositional formulas $\alpha = \alpha_1, \ldots, \alpha_n$ such that for each quantification level k, the formula α_k limits the possible moves for the player at quantification level k. Concrete values of variables are obtained by a SAT call on α_k. At the same time, any newly decided value is propagated in the original formula. The algorithm therefore alternates between QBF-style propagation and SAT solving and invokes learning whenever a dead-end is reached. Since the formulas α_k over-approximate the possible moves of the player at the level k, we refer to these formulas as *abstractions* and to their strengthening as *refinement*.

The pseudocode is presented in Algorithm 1. Initially the vector of abstractions α sets all α_k to true, thus allowing any moves. The algorithm iterates until one of the players is unable to recover from a loss. From the hybrid nature of the algorithm, a loss might happen in two places: if propagation reaches a conflict, or, one of the α_k becomes unsatisfiable. We discuss these situations in more detail later on.

At the beginning of each iteration, QBF propagation is performed on the input formula Φ and the set of current decisions δ (see Section 2.3.1 for `Propagate`). If this propagation is successful, it gives us the inferred literals τ and we move to decision making. If the propagation leads to a conflict for a player loser, conflict resolution is invoked. For this purpose we invoke the subroutine `LearnAndRefine`, which either returns a quantification level where to backtrack or returns $\bot$, meaning that the player loser cannot recover from the conflict. If the player cannot recover from a conflict, it means that the formula is solved because at this point we're sure there is no winning strategy for that player—in particular, the formula is true iff $\mathsf{loser} = \forall$.

If propagation does not yield a conflict, the algorithm calculates the quantification level k for which a decision needs to be made. This level is determined as the outermost level for which there exists some unassigned variable. Hence, at this point, any variable at a quantification level $< k$ was given a value by either a decision or propagation. Given all these current values τ, a SAT solver is invoked on α_k (line 12). If the SAT solver yields satisfiability, the obtained satisfying assignment μ is used to make a new decision. Otherwise, the routine `ResolveUnsat` is invoked. The routine ensures that if the same decisions were to be repeated in the future, the propagator will derive a conflict.

Before we discuss the routines `LearnAndRefine` and `ResolveUnsat`, let us look at some global properties of the algorithm. The following definition tells us what is a correct abstraction. Intuitively, the abstractions must be such that if the current decisions make α_k unsatisfiable, then the player Q_k can no longer win under those decisions.

Definition 2 (abstraction)**.** *Let Z_k denote the free variables of α_k except for the variables $X_1, \ldots, X_k$, i.e. $Z_k \triangleq \mathtt{free}(\alpha_k) \setminus \bigcup_{i=1}^{k} X_i$. Consider the formula $Q_1 X_1 \ldots Q_k X_k \ldots . \phi$.*

We say that α_k is an abstraction for level k *if the following conditions are satisfied:*

1. *If $Q_k = \exists$, then $(Q_{k+1} X_{k+1} \ldots . \phi)$ implies $(\exists Z_k . \alpha_k)$.*

2. *If $Q_k = \forall$, then $\neg(Q_{k+1} X_{k+1} \ldots . \phi)$ implies $(\exists Z_k . \alpha_k)$.*

In the subsequent sections we will look at the implementation of `LearnAndRefine` and `ResolveUnsat`—the two routines that handle a loss of one of the players. However, Algorithm 1 can be seen as a template for a family of algorithms with these two routines implemented differently. For the algorithm to be sound, the following requirements must be fulfilled. The first requirement asserts that the abstraction vector $\alpha = \alpha_1, \ldots, \alpha_n$ consists of correct abstractions (see Definition 2).

Algorithm 2: Learning and refining from a conflict

```
1 Function LearnAndRefine($\delta, \Phi, \alpha$, loser)
  input  : A set of decisions $\delta$ that leads to the loss of the player loser. The formula $\Phi$ and
           abstraction $\alpha$.
  output: Quantification level or $\bot$, the updated abstraction $\alpha$ and $\Phi$.
2 begin
3     $C \leftarrow$ Learn(loser, $\delta, \Phi$)                                // clause learning
4     $k \leftarrow \mathsf{qlevel}_{\mathsf{loser}}(C)$
5     if $k = \bot$ then return $\bot$
6     $\alpha_k \leftarrow$ Refine($\alpha_k$, loser, $C$)
7     $\Phi \leftarrow$ StrengthenPropagator($\Phi$, loser, $C$)
8     return $(k, \alpha, \Phi)$
```

Requirement 1. *Each abstraction α_k is updated using the function* `Refine` *and we require that this function maintains that α_k is an abstraction.*

The second requirement asserts that `ResolveUnsat` does not lose any possible solutions to the formula.

Requirement 2. *If $\Phi' =$* `ResolveUnsat`*(τ, Q_k, Φ) then both players have the same sets of winning strategies for Φ and Φ'.*

The above requirements ensure the following invariants.

Invariant 1. *Algorithm 1 preserves the invariant that each α_k is an abstraction for the quantification level k.*

Invariant 2. *Let Φ_0 be the input formula to Algorithm 1. Then Φ and Φ_0 have the same sets of winning strategies for both of the players.*

3.1 LearnAndRefine

Algorithm 2 details the `LearnAndRefine` routine, which is used whenever a conflict is reached. It first invokes conflict analysis, which returns a learned clause C (see Section 2.3.3 for `Learn`). The backtracking quantification level k is then computed as the inner-most level of loser in C. If C does not contain any literals of loser, it means that the conflict is unresolvable. The reason for that is that the opposing player can always falsify C (a special case of this is $C = \bot$). Otherwise, the abstraction α_k is refined based on the clause C, using the function `Refine`. This refinement lies in the core of the algorithm and is discussed in greater detail in subsequent section 3.3. Finally, the learned clause is inserted into the propagator (see Section 2.3.2 for `StrengthenPropagator`).

3.2 ResolveUnsat

`ResolveUnsat`, shown in Algorithm 3, is invoked when an abstraction is unsatisfiable under the current decisions. The parameter τ', obtained from the SAT solver, is a subset of facts responsible for the unsatisfiability. Hence, τ' represents a *no-good* for the player loser. Analogously to the previous routine, the quantification backtracking level is computed as the innermost level of literals of loser in τ'. As before, if τ' has no literals of loser, then the situation cannot be

Algorithm 3: Analysing an unsatisfiable abstraction.

```
1 Function ResolveUnsat($\tau'$, loser, $\Phi$)
  input  : Set of decisions $\tau'$, losing player loser, formula $\Phi$.
  output: Updated formula $\Phi$ or $\bot$.
2 begin
3     $k \leftarrow \mathsf{qlevel}_{\mathsf{loser}}(\tau')$
4     if $k = \bot$ then return $\bot$                                    // no recovery
5     $\delta \leftarrow \emptyset$                                               // reset propagator
6     $C \leftarrow \{\bar{\ell} \mid \ell \in \tau', \mathsf{qlevel}(\ell) \leq k\}$          // blocking clause from core
7     return StrengthenPropagator($\Phi$, loser, $C$)   // strengthening propag. for loser
```

resolved. Otherwise, the propagator is strengthened by a blocking clause constructed from τ'. Observe that any literals of τ' at levels inner to k can be ignored because the opponent can always choose to satisfy those if the rest is already true.

Proposition 1 (blocking soundness)**.** *The clause C constructed on line 6 of Algorithm 3 is such that if $\Phi' =$ `StrengthenPropagator`(Φ, Q_k, C) then the player Q_k has the same set of winning strategies for Φ' as for Φ.*

The upcoming sections are devoted to the implementation of the function `Refine`, whose objective is to strengthen the abstractions based on the conflict.

3.3 Refine

We present the function `Refine` for the existential player as it is symmetrical for the universal case. Refinement of α_k takes place when there is a conflict in the propagator and QBF learning learnt a clauses C for which $\mathsf{qlevel}_\exists(C) = k$. `Refine` has two principal ingredients:

Conflict-based Strategy Extraction Compute a strategy $\mathscr{S}$ for the opposing player $\forall$, based on the proof of C.

Abstraction Refinement Based on $\mathscr{S}$, strengthen α_k so that the player $\exists$ is not beaten by the same strategy in the future.

We first describe our approach to abstraction refinement, which relies on a strategy.

3.3.1 Abstraction Refinement

Let us suppose that we are given a strategy $\mathscr{S}$ that contains strategies for those and only those universal variables that are at a higher quantification level than k. Given the original matrix ϕ of that input formula, define refinement of α_k as

$$\alpha_k \wedge \phi|_\xi \wedge \textstyle\bigwedge_{\mathrm{S}_z \in \mathscr{S}} z' = \mathrm{S}_z, \text{ where } \xi = \{z'/z \mid \mathsf{qlevel}(z) > k\} \text{ and } z' \text{ is a fresh copy of } z. \quad (1)$$

The refinement strengthens the existing α_k with the matrix ϕ where all variables inner to k are freshened, and, the opponent's variables are set to the values dictated by the strategy. The intuition behind this is that we eliminate all the universal variables by fixing the strategy for them. This means that any move of the existential player determined by α_k must be such that the universal player can no longer counter with the strategy in question.

$$\frac{}{C}\ \text{(Axiom)} \qquad\qquad \frac{C_1 \lor x \qquad C_2 \lor \neg x}{C_1 \lor C_2}\ \text{(LDQ-Res)}$$

C is a clause in the matrix. Variable x is existential. If there is some universal literal ℓ s.t. $\ell \in C_1$ and $\neg\ell \in C_2$, then $\mathsf{qlevel}(x) < \mathsf{qlevel}(\ell)$ (cf. [1]).

Figure 1: The rules of Q^w-resolution.

This type of refinement is inspired by the approach of RAReQS [15, 14]. While in RAReQS, only the universal variables at the quantification level $k+1$ are set, strategies enable setting multiple levels at the same time. This enables maintaining the flat architecture of the solver—whereas recursive calls are needed in RAReQS.
The following proposition ensures that the refinement given by equation (1) gives us a correct abstraction (see Definition 2).

Proposition 2. *Consider the formula $Q_1 X_1 \ldots \exists X_k \forall X_{k+1} \ldots .\, \phi$, and $\mathscr{S}$ a set of strategies for the variables $X_{k+1}, X_{k+3}, \ldots$. The following formula is an abstraction for level k:*

$$\phi|_\xi \land \bigwedge_{\mathrm{S}_z \in \mathscr{S}} z' = \mathrm{S}_z \quad \textit{where } \xi = \{z'/z \mid \mathsf{qlevel}(z) > k\} \tag{2}$$

Proof. The strategies $\mathrm{S}_x \in \mathscr{S}$ represent interpretations of the Herbrand functions for $\Phi_\tau = (\forall X_{k+1} \ldots .\, \phi|_\tau)$. If Φ_τ is true, however, then the formula (2) must be satisfiable for any such interpretations (cf. [17]).

□

Corollary 2. *The formula (1) is an abstraction for level k.*

Remark 1. *To realize refinement for $\forall$ as the losing player, equation (1) uses $\neg\phi$ rather than ϕ. Analogously, proof of Proposition 2 is adapted to Skolem, rather than Herbrand functions.*

3.3.2 Conflict-based Strategy Extraction

The main idea for building a strategy after a propagator reaches a conflict is to look at the proof of the learned clause and construct a strategy from the said proof. Known approaches exist for constructing strategies from refutations [1, 13, 2]. We will essentially generalize this for proofs of arbitrary clauses, rather than just the empty clause. At the same time, we will maintain certain properties necessary for the overall functioning of the algorithm.

A number of proof systems exist that correspond to QBF solving. For the purpose of this work, we use a calculus that is a modification of the existing ones and is more suitable for our purposes.

Definition 3 (Q^w-resolution)**.** *The Q^w-resolution calculus is defined by the rules in Figure 1. A sequence of clauses $C_1, \ldots, C_n$ is a* Q^w-resolution proof *of a clause C, if $C = C_n$ and each clause either belongs to the original matrix or is derived from previous clauses by a LDQ-resolution step. A Q^w-resolution proof is a* refutation *if it is a proof of a clause C that contains only universal variables.*

$$\frac{C \vee u \in \phi}{C \vee u : (u \leftarrow 0)} \qquad \frac{C \vee \bar{u} \in \phi}{C \vee \bar{u} : (u \leftarrow 1)} \qquad \frac{C \in \phi}{C : (u \leftarrow \star)}\ u, \bar{u} \notin C$$

$$\frac{C_1 \vee x : (u \leftarrow f_1) \qquad C_2 \vee \bar{x} : (u \leftarrow f_2)}{C_1 \vee C_2 : (u \leftarrow x\,?\,f_2\ :\ f_1)}$$

We use ϕ for a CNF matrix. Variable x is existential. Furthermore, we simplify realizer expressions as follows:

$$\begin{aligned} x\,?\,\star\ :\ f &\ \rightarrow\ f \\ x\,?\,f\ :\ \star &\ \rightarrow\ f \\ x\,?\,f\ :\ f &\ \rightarrow\ f \end{aligned}$$

Figure 2: `Build` function: extracting strategies from Q^w resolution proofs

For the remainder of the paper we always assume that axiom clauses are non-tautologous.

Q^w-resolution enables us to apply standard clause learning [25] in the QBF setting. As usual, after a conflict has been reached through propagation, resolution steps are applied in the inverse order of the propagation [30].

For a variable u, we build a strategy using a function $\texttt{Build}(u, \pi)$, which returns a strategy for u based on a Q^w-proof π. This strategy might be the special symbol $\star$, meaning that π does not impose any constraints on u. In the following we abuse the notation by writing $\texttt{Build}(u, C)$, meaning that `Build` is invoked on a proof of C. The function is defined by the rules of Figure 2. For the axiom clauses, it makes sure that the strategy falsifies the pertaining literal. For clauses obtained by a resolution step, the following scenarios may be identified. If both antecedents give the same strategy for u, that strategy is considered. If one of the antecedents does not impose a strategy on u, then it is ignored. The last scenario is the most interesting one. If the two antecedents yield different strategies for u, an if-then-else expression is built, splitting on the pivot variable. The following lemma shows that the non-trivial strategies are obtained only when C contains u with complementary signs.

Lemma 1. *Let* $\mathrm{S}_u = \texttt{Build}(u, C)$*. If* $u \in C$*,* $\bar{u} \notin C$*, then* $\mathrm{S}_u = 0$ *and analogously, if* $\bar{u} \in C$*,* $u \notin C$*, then* $\mathrm{S}_u = 1$*. If* $u, \bar{u} \notin C$*, then* $\mathrm{S}_u = \star$*.*

The following lemma shows that the non-trivial scenario only takes place when $\mathsf{qlevel}(e) < \mathsf{qlevel}(u)$, which ensures that the function `Build` returns a strategy for u.

Proposition 3. $\texttt{Build}(u, C)$ *is a strategy for* u*.*

Proof. (By induction on the derivation of C*.)* If C is an axiom, $\texttt{Build}(u, C)$ is trivially a strategy for u, since it's a constant function.

From Lemma 1, if the if-the-else expression is created and not simplified, it must be that u is in one of the antecedents and $\bar{u}$ in the other. According to the rules of Q^w, it must be that $\mathsf{qlevel}(e) < \mathsf{qlevel}(u)$. Hence, the constructed if-then-else expression is a strategy for u. □

Proposition 4. *Let* U *be a set of universal variables and* C *some clause derived by* Q^w*-resolution proof* π*. Define* C' *as* C *where all of literals on* U *are removed and define the substitution* $\tau_D = \{\texttt{Build}(u, D)/u \mid \texttt{Build}(u, D) \neq \star\}$ *for any clause* $D \in \pi$*. Then* $\phi|_{\tau_C} \Rightarrow C'$*.*

Algorithm 4: Refine an abstraction α_k

```
1 Function Refine($\alpha_k$, loser, $C$)
  input  : An abstraction $\alpha_k$, learned clause $C$ for the player loser.
  output: Updated $\alpha_k$.
2 begin
3     $\mathscr{S} \leftarrow \left\{ \mathrm{S}_x \triangleq \mathtt{Build}(x, C) \mid Q_{\mathsf{qlevel}(x)} \neq \mathsf{loser}, \mathsf{qlevel}(x) > k, \mathtt{Build}(x, C) \neq \star \right\}$
4     $\mathscr{S} \leftarrow \mathscr{S} \cup \left\{ \mathrm{S}_x \triangleq \mathsf{false} \mid Q_{\mathsf{qlevel}(x)} \neq \mathsf{loser}, \mathsf{qlevel}(x) > k, \mathtt{Build}(x, C) = \star \right\}$
5     $\xi \leftarrow \{z'/z \mid \mathsf{qlevel}(z) > k\}$              // compute renaming to fresh variables
6     $\psi \leftarrow \mathsf{loser} = \exists\, ?\, \phi \,:\, \neg\phi$                  // negate matrix if necessary
7     return $\alpha_k \wedge \psi|_\xi \wedge \bigwedge_{\mathrm{S}_x \in \mathscr{S}} x' = \mathrm{S}_x$
```

Proof. (By induction on the derivation of C.) If C is an axiom, then the implication holds trivially because $\phi \Rightarrow C$ and because we assume non-tautologous axioms.

To show the hypothesis for a resolution step of $C_1 \vee e$ and $C_2 \vee \bar{e}$, with e as the pivot variable, we need to show that for any assignment μ such that $\mu \models \phi|_{\tau_C}$, it holds that $\mu \models C'$. Without loss of generality $\mu(e) = 0$. Then it needs to be shown that $\phi|_{\tau_C} \Rightarrow \phi|_{\tau_{C_1}}$. This is easily verified by showing that $\tau_{C_1} \cup \{0/e\}$ is more restrictive than $\tau_C \cup \{0/e\}$. □

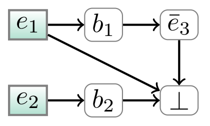
Algorithm 4, summarises the content of the last two sections into the implementation of the routine `Refine`. The function `Build` is used to construct strategies for the opponent variables inner to the quantification level k. Any variables that are not constrained by `Build`, i.e. $\mathtt{Build}(x, C) = \star$, are given a default value. The variables inner to k are all freshened and additionally the opponent variables are fixed to the value set by the computed strategy. This operation is performed on the matrix ϕ if the loser is the existential player and on the negation of ϕ if the loser is the universal player.

3.4 Good Clauses for Strategies

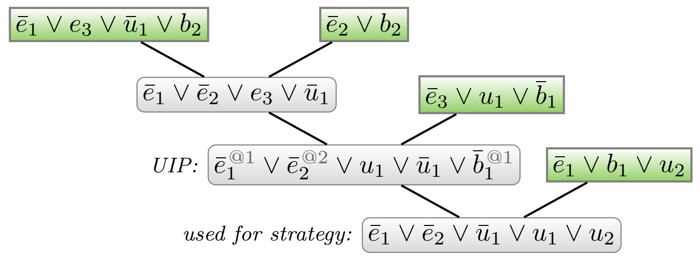
An important property of clause learning in SAT as well as QBF is that any learned clause C is *asserting*. This effectively means that 1) there is a unique literal $\ell \in C$ that is currently false and is at a decision level k; 2) such that if decisions are retracted up to level k, the value of ℓ is forced to true by propagation. We refer to such ℓ as the *asserting literal* of C.

However, if we wish to carry this behaviour over by refinement, we cannot rely on an arbitrary learned clause. The issue is that the asserting literal may not be the literal at the highest quantification level. So for instance, for the prefix $\ldots \exists e \forall u \exists b \ldots$, the clause $(e \vee u \vee b)$ *can* be a learned clause with e asserting, as long as b is propagated before e. Such learned clause does not guarantee a good strategy. If this clause is used to refine $\alpha_{\mathsf{qlevel}(b)}$, no new restriction is imposed on the value of e since that is decided by models of $\alpha_{\mathsf{qlevel}(e)}$. If this clause is used to refine $\alpha_{\mathsf{qlevel}(e)}$, Proposition 4 only guarantees that the refined $\alpha_{\mathsf{qlevel}(e)}$ implies $e \vee u \vee b'$, where b' is a fresh copy of b, but no guarantee that b' is forced to 0 and hence not guarantee that the refinement forces $e = 1$.

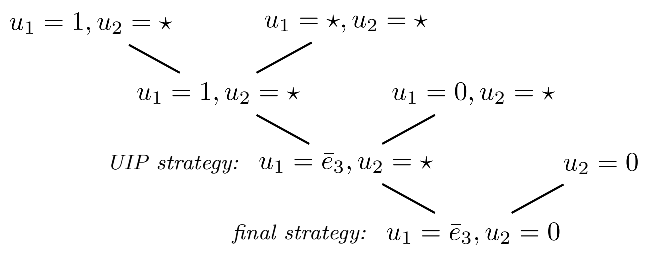
The solution we propose is to construct strategies only from learned clauses where the asserting literal is at the highest qualification level with respect to the rest of the existential literals in the clause. This is achieved as follows. Perform clause learning by a sequence of resolution steps in the reverse order of propagation, as usual. Once an asserting clause is derived, record the quantification level k of the asserting literal. Continue resolution until there



(a) implication graph


$\bar{e}_1 \vee e_3 \vee \bar{u}_1 \vee b_2$
$\bar{e}_2 \vee b_2$
$\bar{e}_1 \vee \bar{e}_2 \vee e_3 \vee \bar{u}_1$
$\bar{e}_3 \vee u_1 \vee \bar{b}_1$
UIP: $\bar{e}_1^{@1} \vee \bar{e}_2^{@2} \vee u_1 \vee \bar{u}_1 \vee \bar{b}_1^{@1}$
$\bar{e}_1 \vee b_1 \vee u_2$
used for strategy: $\bar{e}_1 \vee \bar{e}_2 \vee \bar{u}_1 \vee u_1 \vee u_2$

(b) learning


$u_1 = 1, u_2 = \star$
$u_1 = \star, u_2 = \star$
$u_1 = 1, u_2 = \star$
$u_1 = 0, u_2 = \star$
UIP strategy: $u_1 = \bar{e}_3, u_2 = \star$
$u_2 = 0$
final strategy: $u_1 = \bar{e}_3, u_2 = 0$

(c) strategy

are no existential literals with a higher qualification level than k in the clause. Additionally, if the learned clause contains any Tseitin variables, resolution continues as well. This approach is illustrated by the following example.

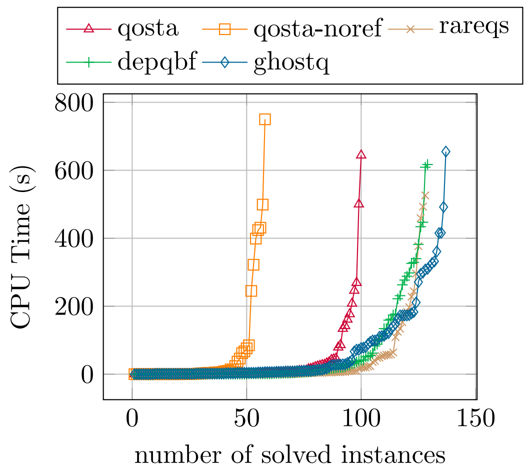
Example 2. *Consider:* $\exists e_1 e_2 e_3 \forall u_1 \exists b_1 \forall u_2 \exists b_2. (\bar{e}_1 \vee b_1 \vee u_2) \wedge (\bar{e}_2 \vee b_2) \wedge (\bar{e}_3 \vee u_1 \vee \bar{b}_1) \wedge (\bar{e}_1 \vee e_3 \vee \bar{u}_1 \vee b_2)$

Figure 3a shows an implication graph, with $e_1 = e_1 = 1$ *as decisions, Figure 3b the corresponding resolution proof, and Figure 3c the strategies corresponding to each clause.*

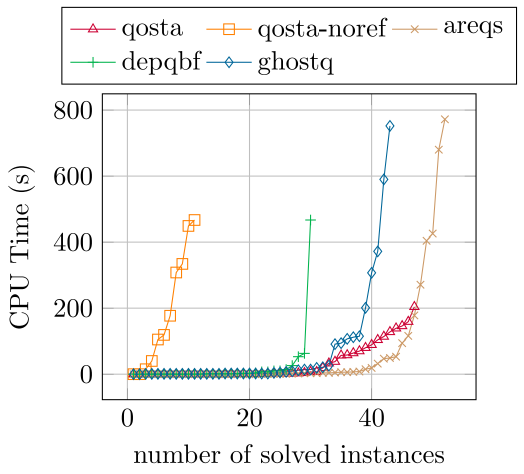
For the UIP clause $\bar{e}_1^{@1} \vee \bar{e}_2^{@2} \vee u_1 \vee \bar{u}_1 \vee \bar{b}_1^{@1}$ *we have the asserting literal* $\bar{e}_2$*. At the same time, however, the literal with the highest qualification level is* $\bar{b}_1$*. If, at this point we would refine* α_1 *based off the strategy* $\mathscr{S} = \{\mathrm{S}_{u_1}(e_1, e_2, e_3) = \bar{e}_3, \mathrm{S}_{u_2}(e_1, e_2, e_3, b_1) = \star\}$*, we would only guarantee* $\alpha_1 \models \bar{e}_1 \vee \bar{e}_2 \vee \bar{b}'_1$*, where* b'_1 *is a fresh copy of* b_1*, due to Proposition 4. In the propagator, after adding the UIP, deciding* $e_1 = 1$ *leads to* $e_2 = 0$*, while the refined* α_1 *would not have this property. Therefore, we do not stop the learning procedure at the UIP, and continue resolving, until the asserting literal becomes the literal with the highest quantification level. This gives us the strategy* $\mathscr{S} = \{\mathrm{S}_{u_1}(e_1, e_2, e_3) = \bar{e}_3, \mathrm{S}_{u_2}(e_1, e_2, e_3, b_1) = 0\}$*, guaranteeing* $\alpha_1 \models \bar{e}_1 \vee \bar{e}_2$*, again, due to Proposition 4.*

4 Experimental Evaluation

We have implemented a prototype based on Algorithm 1, named QOSTA—(*Qbf sOlver with Strategy exTrAction*). Since the implementation requires a tight integration between propagation and the main solver, we have decided to implement the QBF propagator from scratch. MiniSAT 2.2 was used as the underlying SAT solver [9]. The experiments were carried out on an Intel Xeon 5160 3GHz with 4GB of memory. The time limit was set to 800 seconds and the memory limit to 2GB. All instances were preprocessed by bloqqer [8] (except for GhostQ, which was given unpreprocessed inputs, since GhostQ has difficulty handling the output of bloqqer).



(a) Eval '12



(b) 2QBF '10

Figure 3: Cactus plots for the experiments.

To evaluate the effect of refinement, we considered two versions of the solver: one as described in Algorithm 1–QOSTA and the second without the refinement—QOSTA-NOREF. QOSTA-NOREF corresponds to Algorithm 1 without line 6, and effectively evaluates our QBF propagator.

Other solvers included in the evaluation are the non-CNF-based solver GhostQ [19], CDCL-style solver with variable dependency computation DepQBF [21], the CEGAR-based solvers RAREQS and AREQS [15, 14].

Figure 3 shows cactus plots for two benchmark families. Figure 3a is the QBF evaluation 2012 benchmark set.[1] Figure 3b the 2QBF[2] evaluation 2010 benchmark set.[3] QOSTA does not perform well on the general set of benchmarks but it is the second best on the 2QBF family. The version without refinement (QOSTA-NOREF) is significantly worse than the general version and all the other solvers. This indicates that the refinement is essential to the algorithm's workings but also that there is a lot of space for improvement in the implementation.

5 Related Work

There are two distinctive strands in QBF solving. In one approach, a solver traverses the search space, while pruning it by propagation, similarly to SAT solvers [30, 31, 19, 21, 12, 22]. In the second approach, a solver eliminates (expands) quantifiers obtaining thus a SAT problem [26, 6, 20, 3].

The weak spot of expansion is the exponential blowup in space. This is mitigated by the CEGAR approaches of the solver AREQS and its recursive extension RAREQS, which expand the formula gradually [15, 14]. More recently the solvers QESTO [16], QELL [29], and CAQE [27] build on RAREQS with variations in refinement but also in flat architecture, while RAREQS builds a tree of abstractions. Expansion and search has been shown substantially different from a theoretical perspective via proof complexity [5].

[1]http://qbf.satisfiability.org/gallery/.
[2]By 2QBF is meant instances with 2 levels of qualification.
[3]http://www.qbflib.org/index_eval.php.

6 Summary and Future Work

This paper presents a novel technique for combining DPLL-style solving and abstraction refinement in a QBF solver. Strategy extraction serves as the coupling between the two techniques. Abstractions in the form of propositional formulas are maintained for each quantification level and search-space is traversed while using abstractions for making new decisions. Once a conflict is reached via propagation, a strategy is extracted and used to refine the appropriate abstraction.

The presented techniques open a number of avenues for future work. Other or complementary methods for strategy extraction could be envisioned. For instance, as different strategies are learned throughout solving, new strategies could be devised by combining the existing ones. More QBF techniques should be considered for integration into the solver, such as variable dependencies, pure literal detection or in-processing in general. The constructed strategies were applied for refining abstractions. However, they can be used in different settings. For instance as a prediction for the opposing player in solvers like QESTO. Strategy extraction could be employed in other domains than QBF. In particular, in SMT with quantification.

Acknowledgments

This work was partially supported by the CMU-Portugal grant AMOS (CMUP-EPB/TIC/-0049/2013).

References

[1] V. Balabanov and J.-H. R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

[2] V. Balabanov, J. R. Jiang, M. Janota, and M. Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *Conference on Artificial Intelligence (AAAI)*, pages 3694–3701, 2015.

[3] M. Benedetti. sKizzo: a suite to evaluate and certify QBFs. In *CADE*, 2005.

[4] M. Benedetti and H. Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[5] O. Beyersdorff, L. Chew, and M. Janota. Proof complexity of resolution-based QBF calculi. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76–89. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[6] A. Biere. Resolve and expand. In *SAT*, pages 238–246, 2004.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[8] A. Biere, F. Lonsing, and M. Seidl. Blocked clause elimination for QBF. In *CADE*, pages 101–115, 2011.

[9] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2004.

[10] E. Giunchiglia, P. Marin, and M. Narizzano. Reasoning with quantified boolean formulas. In Biere et al. [7], pages 761–780.

[11] E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res. (JAIR)*, 26:371–416, 2006.

[12] A. Goultiaeva, M. Seidl, and A. Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In *Design, Automation & Test in Europe (DATE)*, pages 811–814, 2013.

[13] A. Goultiaeva, A. Van Gelder, and F. Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In T. Walsh, editor, *IJCAI*, pages 546–553. IJCAI/AAAI, 2011.

[14] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke. Solving QBF with counterexample guided refinement. In *SAT*, pages 114–128, 2012.

[15] M. Janota and J. Marques-Silva. Abstraction-based algorithm for 2QBF. In *SAT*, 2011.

[16] M. Janota and J. Marques-Silva. Solving QBF by clause selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[17] H. Kleine Büning, K. Subramani, and X. Zhao. Boolean functions as models for quantified boolean formulas. *J. Autom. Reasoning*, 39(1):49–75, 2007.

[18] W. Klieber. *Formal Verification Using Quantified Boolean Formulas (QBF)*. PhD thesis, Carnegie Mellon University, 2014.

[19] W. Klieber, S. Sapra, S. Gao, and E. M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *SAT*, 2010.

[20] F. Lonsing and A. Biere. Nenofex: Expanding NNF for QBF solving. In *SAT*, pages 196–210, 2008.

[21] F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver. *JSAT*, 7(2-3):71–76, 2010.

[22] F. Lonsing, U. Egly, and A. Van Gelder. Efficient clause learning for quantified boolean formulas via QBF pseudo unit propagation. In *SAT*, pages 100–115, 2013.

[23] F. Lonsing, M. Seidl, and A. Van Gelder. The QBF gallery: Behind the scenes. *CoRR*, abs/1508.01045, 2015.

[24] J. P. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [7], pages 131–153.

[25] J. P. Marques Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[26] G. Pan and M. Y. Vardi. Symbolic decision procedures for QBF. In *CP*, pages 453–467, 2004.

[27] M. N. Rabe and L. Tentrup. CAQE: A certifying QBF solver. In *FMCAD*, 2015. `http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD15/papers/paper50.pdf`.

[28] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, Part II, ed. A.O. Slisenko, 1968.

[29] K.-H. Tu, T.-C. Hsu, and J.-H. R. Jiang. QELL: QBF reasoning with extended clause learning and levelized SAT solving. In *SAT*, pages 343–359, 2015.

[30] L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *International Conference on Computer-Aided Design (ICCAD)*, pages 442–449, 2002.

[31] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *CP*, pages 200–215. Springer, 2002.