Interpolation-Based Semantic Gate Extraction and Its Applications to QBF Preprocessing

Friedrich Slivovsky
TU Wien, Vienna, Austria
fs@ac.tuwien.ac.at

Abstract. We present a new semantic gate extraction technique for propositional formulas based on interpolation. While known gate detection methods are incomplete and rely on pattern matching or simple semantic conditions, this approach can detect any definition entailed by an input formula.

As an application, we consider the problem of computing unique strategy functions from Quantified Boolean Formulas (QBFs) and Dependency Quantified Boolean Formulas (DQBFs). Experiments with a prototype implementation demonstrate that functions can be efficiently extracted from formulas in standard benchmark sets, and that many of these definitions remain undetected by syntactic gate detection.

We turn this into a preprocessing technique by substituting unique strategy functions for input variables and test solver performance on the resulting instances. Compared to syntactic gate detection, we see a significant increase in the number of solved QBF instances, as well as a modest increase for DQBF instances.

1 Introduction

Due to the effectiveness of modern satisfiability (SAT) solvers [20], propositional logic has become the language of choice for encoding hard combinatorial problems arising in areas such as electronic design automation [50] and AI planning. Since many of these problems are hard for levels of the polynomial hierarchy beyond NP, their propositional encodings can be exponentially larger than their original descriptions. This imposes a limit on the problem instances that can be feasibly solved even with extremely efficient SAT solvers, and has prompted research on decision procedures for more succinct logical formalisms such as Quantified Boolean Formulas (QBFs).

Quantified Boolean Formulas (QBFs) are propositional formulas combined with universal and existential quantification over truth values and offer much more succinct encodings of problems from domains such as planning and synthesis [12]. At the same time, QBF evaluation is PSPACE-complete, and in spite

This research was supported by the Vienna Science and Technology Fund (WWTF) under grant number ICT19-060.
of substantial progress in solver technology, many practically relevant instances remain hard to solve.

In part, this hardness appears to be a matter of encoding. The most commonly used format for QBFs is Prenex Conjunctive Normal Form (PCNF). A PCNF formula consists of a quantifier prefix and a matrix in conjunctive normal form. As in the case of propositional logic, any QBF can be converted to PCNF with linear overhead but this transformation is known to adversely affect solver performance [1]. This appears to be due to two issues: First, conversion to CNF causes a bias towards reasoning about unsatisfiability while making it difficult to reason about solutions, violating the inherent duality of QBF solving. Second, prenexing introduces spurious variable dependencies that needlessly constrain solvers [5,40]. In light of these issues, researchers have introduced two new formats for representing non-CNF (and even non-prenex) QBFs in the QCIR [30] and QAIGER standards, and solvers supporting these standards have been developed. When only a PCNF encoding is available, gate extraction techniques can be used to (re)construct a non-CNF QBF [21]. Syntactic gate extraction relies on the detection of patterns of clauses and auxiliary variables introduced when converting a propositional formula to CNF [16]. The corresponding algorithms are fast but incomplete and can only detect definitions from a pre-defined library of gates.

In this paper, we introduce a new semantic gate extraction technique based on SAT solving and interpolation. In contrast to known approaches, this method is complete: a definition $\psi$ of a variable $x$ can be extracted from a propositional formula $\varphi$ whenever the equivalence $x \equiv \psi$ is entailed by $\varphi$. We obtain this result as a generalization of recent work that leverages definability for propositional model counting [25,33]. Owing to a result known as Padoa’s Theorem, determining whether a variable $x$ is definable in terms of $X$ is in coNP and can be decided by a SAT call [33]. We show that a definition $\psi$ of $x$ in terms of $X$ can be obtained as an interpolant of the formula passed to the SAT solver (Theorem 2). For SAT solvers that use a proof system with feasible interpolation—in particular, CDCL solvers that generate resolution proofs [32]—this means a definition can be efficiently extracted from a proof of definability.

We apply this new gate extraction technique to identify unique strategy functions of QBFs and Dependency QBFs. In a controller synthesis setting, a variable with a unique strategy function corresponds to a control signal with a unique (as a Boolean function) implementation. We can add such an implementation to the specification without affecting the remaining control signals.

Experiments with a prototype show that definitions can be efficiently computed for formulas from standard QBF benchmark sets, and that for many instances a large fraction of variables have unique strategy functions that cannot be identified by syntactic gate detection. We further test the performance of solvers on instances obtained by replacing input variables with their definitions. For 2QBF formulas and PCNF formulas, this significantly increases the number of instances solved by some systems compared to purely syntactic gate extraction. Our experiments further show that semantic gate detection is orthogonal to techniques implemented in state-of-the-art preprocessors.
Semantic gate detection is efficient and conceptually simple. By definition, it preserves logical equivalence and is compatible with strategy extraction. As such, we believe it is an essential addition to the state of the art in preprocessing (D)QBF.

2 Preliminaries

We assume a countably infinite set $V$ of propositional variables and consider propositional formulas constructed from $V$ using the connectives $\neg$ (negation), $\land$ (conjunction), $\lor$ (disjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (the biconditional). For a propositional formula $\varphi$, we write $\text{var}(\varphi)$ to denote the set of variables occurring in $\varphi$. A literal is a variable $v$ or a negated variable $\neg v$. A clause is a finite disjunction of literals. A clause is tautological if it contains both $v$ and $\neg v$ for some variable $v$. A propositional formula is in conjunctive normal form (CNF) if it is a finite conjunction of non-tautological clauses. An assignment of a subset $X \subseteq V$ of variables is a function that maps $X$ to the set $\{0, 1\}$ of truth values. For a set $X$ of variables we let $[X]$ denote the set of assignments of $X$. Two assignments $\sigma : X \rightarrow \{0, 1\}$ and $\tau : Y \rightarrow \{0, 1\}$ agree on a subset $W \subseteq X \cap Y$ of their common domain if $\sigma(w) = \tau(w)$ for each $w \in W$. For two assignments $\sigma : X \rightarrow \{0, 1\}$ and $\tau : Y \rightarrow \{0, 1\}$ that agree on the entire intersection of their domains we define the combined assignment $\sigma \cup \tau : X \cup Y \rightarrow \{0, 1\}$ as $(\sigma \cup \tau)(v) = \sigma(v)$ if $v \in X$ and $(\sigma \cup \tau)(v) = \tau(v)$ otherwise.

For a propositional formula $\varphi$ and an assignment $\tau : X \rightarrow \{0, 1\}$ with $\text{var}(\varphi) \subseteq X$, we let $\varphi[\tau]$ denote the truth value obtained by evaluating $\varphi$ under $\tau$. The formula $\varphi$ is satisfied by $\tau$ if $\varphi[\tau] = 1$. In this case we call $\tau$ a satisfying assignment of $\varphi$. Otherwise, if $\varphi[\tau] = 0$, formula $\varphi$ is falsified by $\tau$. A formula is satisfiable if it has a satisfiable assignment, otherwise it is unsatisfiable. A formula $\varphi$ implies a formula $\psi$ if $\varphi \land \neg \psi$ is unsatisfiable.

We consider Quantified Boolean Formulas (QBFs) in Prenex Normal Form (PNF). A QBF $\Phi = Q.\varphi$ in PNF consists of a quantifier prefix $Q$ and a propositional formula $\varphi$, called the matrix of $\Phi$. The quantifier prefix is a sequence $Q_1 x_1 \ldots Q_n x_n$ where $Q_i \in \{\forall, \exists\}$ and the $x_i$ are pairwise distinct variables for $1 \leq i \leq n$. The quantifier prefix defines an ordering $<_\Phi$ on its variables as $x_i <_\Phi x_j$ for $1 \leq i < j \leq n$. We assume that QBFs do not contain free variables and every variable in the quantifier prefix appears in the matrix, formally $\{x_1, \ldots, x_n\} = \text{var}(\varphi)$. Accordingly, we write $\text{var}(\Phi) = \text{var}(\varphi)$ for the set of variables appearing in the QBF $\Phi$. We further assume that every variable of $\Phi$ occurs exactly once in its quantified prefix. The set of existential variables of $\Phi$ is $\text{var}_\exists(\Phi) = \{ x_i \mid 1 \leq i \leq n, Q_i = \exists \}$, and the set of universal variables of $\Phi$ is $\text{var}_\forall(\Phi) = \{ x_i \mid 1 \leq i \leq n, Q_i = \forall \}$. For a variable $x \in \text{var}(\Phi)$, we let $\text{type}_\Phi(x) = Q$ if $x \in \text{var}_Q(\Phi)$, for $Q \in \{\forall, \exists\}$, omitting $\Phi$ from the subscript if the QBF is understood.

Let $\Phi$ a QBF and let $x \in \text{var}(\Phi)$ be one of its variables with $\text{type}(x) = Q$. A strategy function for $x$ is a function $f : [\text{var}(\Phi) \setminus \text{var}_Q(\Phi)] \rightarrow \{0, 1\}$ such that $f(\tau) = f(\tau')$ for any two assignments $\tau$ and $\tau'$ that agree on variables in
\{ v \in \text{var}(\Phi) \setminus \text{var}_Q(\Phi) \mid v <_\Phi x \}\). Given an indexed family \( F = \{ f_x \}_{x \in X} \) of strategy functions such that \( X \subseteq \text{var}(\Phi) \) for \( Q \in \{ \forall, \exists \} \), the response of \( F \) to an assignment \( \tau : (\text{var}(\Phi) \setminus \text{var}_Q(\Phi)) \to \{0, 1\} \) is the assignment \( F(\tau) : X \to \{0, 1\} \) given by \( F(\tau)(x) = f_x(\tau) \). An existential winning strategy (for \( \Phi \)) is a family \( F = \{ f_u \}_{u \in \text{var}_\exists(\Phi)} \) of strategy functions such that, for any universal assignment \( \tau : \text{var}_\forall(\Phi) \to \{0, 1\} \), the assignment \( \tau \cup F(\tau) \) satisfies the matrix of \( \Phi \). Dually, a universal winning strategy (for \( \Phi \)) is a family \( F = \{ f_u \}_{u \in \text{var}_\forall(\Phi)} \) of strategy functions such that, for any existential assignment \( \sigma : \text{var}_\exists(\Phi) \to \{0, 1\} \), the assignment \( \sigma \cup F(\sigma) \) falsifies the matrix. A QBF \( \Phi \) is true if there is an existential winning strategy for \( \Phi \), and false if there exists a universal winning strategy for \( \Phi \).

3 Semantic Gate Extraction by Interpolation

This work builds on an application of propositional definability to the model counting problem [33]. We begin by recalling two basic concepts.

**Definition 1.** Let \( \varphi \) be a formula, let \( X \) be a subset of its variables, and let \( x \) be a variable. Variable \( x \) is defined in terms of \( X \) in \( \varphi \) if \( \sigma(x) = \tau(x) \) for any two satisfying assignments \( \sigma \) and \( \tau \) of \( \varphi \) that agree on \( X \). A definition of \( x \) by \( X \) in \( \varphi \) is a formula \( \psi \) with \( \text{var}(\psi) \subseteq X \) such that \( \sigma(x) = \psi[\sigma] \) for any satisfying assignment \( \sigma \) of \( \varphi \).

It is readily verified that there is a definition for every variable that is defined. Lagniez et al. [33] observe that the following result can be used to determine whether a variable is defined [34, 39].

**Theorem 1 (Padoa’s Theorem).** Let \( \varphi \) be a formula and let \( X \subseteq \text{var}(\varphi) \) be a subset of its variables. Let \( \varphi' \) be the propositional formula obtained by replacing every variable \( y \in \text{var}(\varphi) \setminus X \) by a new variable \( y' \). Let \( x \in \text{var}(\varphi) \) be a variable. If \( x \notin X \), then \( x \) is defined in \( \varphi \) by \( X \) if, and only if, the formula \( \varphi \land x \land \varphi' \land \neg x' \) is unsatisfiable.

For the purposes of preprocessing in model counting, it is sufficient to know that a variable \( x \) is defined by \( X \) in \( \varphi \), and the above result shows that this can be decided by a SAT solver. It is not necessary to compute the corresponding definition, whose size is not polynomially bounded in the size of \( \varphi \) under common assumptions in computational complexity [33].

While finding definitions is harder than deciding definability in theory, the difference virtually disappears in practice. Our main theoretical contribution, stated as Theorem 2 below, says that a definition can be obtained as an interpolant of the formula constructed in the statement of Padoa’s Theorem. Since interpolants can be efficiently (in linear time) generated from resolution proofs [22, 32], the distinction between detecting definability and computing definitions

---

1 We sometimes refer to existential strategy functions as *Skolem* functions and universal strategy functions as *Herbrand* functions.
becomes moot when a CDCL SAT solver is used to decide (un)satisfiability: once it determines that the formula is unsatisfiable it has already (implicitly or explicitly) produced a proof from which a definition can be extracted at a small overhead.\footnote{Assuming the SAT solver does not use the full power of the DRAT proof system \cite{DRAT}.}

Before proving Theorem 2, we recall the definition of an interpolant following McMillan \cite{McMillan}.

**Definition 2 (Interpolant).** Let $\psi$ and $\chi$ be an formulas such that $\psi \land \chi$ is unsatisfiable. An interpolant for $\psi$ and $\chi$ is a formula $I$ such that

1. $\psi$ implies $I$,
2. $I \land \chi$ is unsatisfiable, and
3. $I$ only refers to variables common to $\psi$ and $\chi$.

Craig’s Interpolation Theorem \cite{Craig} states that every pair of jointly unsatisfiable propositional formulas have an interpolant.\footnote{In fact, the result holds even for first order logic, but we will confine ourselves to the propositional case.} It remains to show that an interpolant for a formula witnessing definability in fact yields a definition.

**Lemma 1.** Let $\varphi$ be a formula and let $X \subseteq \text{var}(\varphi)$ be a subset of its variables. Let $\varphi'$ be the formula obtained by replacing every variable $y \in \text{var}(\varphi) \setminus X$ by a new variable $y'$. For any variable $x \in \text{var}(\varphi) \setminus X$, an interpolant for $\varphi \land x$ and $\varphi' \land \lnot x'$ is a definition of $x$ in $\varphi$.

**Proof.** Let $I$ be an interpolant for $\varphi \land x$ and $\varphi' \land \lnot x'$. By property (3) of Definition 2, $I$ only refers to the common variables $\text{var}(\varphi \land x) \cap \text{var}(\varphi' \land \lnot x') = X$ of these formulas. To see that $I$ defines $x$ in $\varphi$, consider a satisfying assignment $\sigma : \text{var}(\varphi) \rightarrow \{0,1\}$ of $\varphi$. If $\sigma(x) = 1$ then $\varphi \land x$ is satisfied by $\sigma$. The formula $\varphi \land x$ implies $I$ by property (1), so $I[\sigma] = 1$ as well. Otherwise, $\sigma(x) = 0$ and we can construct a satisfying assignment $\sigma'$ of $\varphi' \land \lnot x'$ by setting $\sigma'(v) = \sigma(v)$ for $v \in X$ along with $\sigma'(v') = \sigma(v)$ for $v \in \text{var}(\varphi) \setminus X$. By property (2), $I \land \varphi' \land \lnot x'$ is unsatisfiable, so we must have $I[\sigma'] = I[\sigma] = 0$.

**Theorem 2.** Let $\varphi$ be a formula and let $X \subseteq \text{var}(\varphi)$ be a subset of its variables. Let $\varphi'$ be the formula obtained by replacing every variable $y \in \text{var}(\varphi) \setminus X$ by a new variable $y'$. A variable $x \in \text{var}(\varphi) \setminus X$ is defined in terms of $X$ in $\varphi$ if, and only if, the formula $\varphi \land x \land \varphi' \land \lnot x'$ is unsatisfiable, and a definition of $x$ in terms of $X$ can be obtained as an interpolant for $\varphi \land x$ and $\varphi' \land \lnot x'$.

**Proof.** By Theorem 1 variable $x \in \text{var}(\varphi) \setminus X$ is defined in terms of $X$ in $\varphi$ if, and only if, the formula $\varphi \land x \land \varphi' \land \lnot x'$ is unsatisfiable. Craig’s Interpolation Theorem tells us that in this case there is an interpolant for $\varphi \land x$ and $\varphi' \land \lnot x'$, which defines $x$ in terms of $X$ by Lemma 1.
4 Extracting Unique QBF Strategy Functions

In this section, we show how Theorem 2 can be used to extract unique strategy functions of QBFs. We say that the Skolem (Herbrand) function of an existential (universal) variable $x$ in a QBF is unique if it is the same in every existential (universal) winning strategy. In particular, if $x$ is existentially (universally) quantified and the formula is false (true), then the strategy function of $x$ is trivially unique (there is none). In other words, the strategy function of a variable $x$ is unique if there is at most one such function for $x$ that is part of a winning strategy. The following result states that propositional definability is a sufficient condition for uniqueness of a strategy function.

**Proposition 1.** Let $\Phi = Q_1 x_1 \ldots Q_n x_n. \varphi$ be a QBF. If an existential (universal) variable $x_i$ is defined in terms of variables $X \subseteq \{ x_j \mid 1 \leq j < i, Q_j \neq Q_i \}$ in $\varphi (\neg \varphi)$ its Skolem (Herbrand) function is unique.

**Proof.** We only consider the case where $x_i$ is an existential variable of $\Phi$ (the case where $x_i$ is a universal variable is symmetric). Let $F = \{ f_{x_i} \}_{x_i \in \text{var}_\exists(\Phi)}$ and $G = \{ g_{x_i} \}_{x_i \in \text{var}_\exists(\Phi)}$ be existential winning strategies and $\tau : \text{var}_\forall(\Phi) \rightarrow \{0,1\}$ an assignment to the universal variables. Since $F$ and $G$ are existential winning strategies both $\sigma_F = \tau \cup F(\tau)$ and $\sigma_G = \tau \cup G(\tau)$ must be satisfying assignments of $\varphi$. The assignments $\sigma_F$ and $\sigma_G$ agree on $X \subseteq \text{var}_\forall(\Phi)$, so we must have $f_{x_i}(\tau) = \sigma_F(x_i) = \sigma_G(x_i) = g_{x_i}(\tau)$ because $x_i$ is defined in terms of $X$. Since $\tau$ was chosen arbitrarily, this identity holds for every universal assignment, so the functions $f_{x_i}$ and $g_{x_i}$ coincide.

To see that definability is not a necessary condition for a strategy function to be unique, consider the following example.

**Example 1.** Let $\Phi = \forall x \exists y \forall z. (x \leftrightarrow y) \lor z$. The formula $\psi = x$ represents the unique existential winning strategy (set $y$ to the same value as $x$). However, variable $y$ is not defined in terms of $x$: the assignments $\{x, y, z\}$ and $\{x, \neg y, z\}$ both satisfy the matrix and agree on $x$, but differ on $y$. Intuitively, the reason why the existential strategy function for $y$ is unique in spite of $y$ not being defined is that the universal player would never assign $z$ true as required by one of the assignments witnessing non-definability.

4.1 An Algorithm for Computing Unique Strategy Functions

We now describe an algorithm for computing unique strategy functions of a QBF based on Proposition 1. By using an interpolating SAT solver (ItpSatSolver) that supports both incremental solving and assumptions [22], we can extract definitions for variables of a given quantifier type (universal or existential) using a single solver instance. Pseudocode is shown as Algorithm 1 below.

Let $\Phi = Q_1 x_1 \ldots Q_n x_n. \varphi$ be a QBF and let $Q \in \{ \forall, \exists \}$ be a quantifier type. Algorithm 1 first determines the leftmost variable $x_i$ in the prefix of $\Phi$ that has quantifier type $Q$ (line 3). The strategy function of any variable to the
right of \(x_i\) in the prefix (including \(x_i\) itself) may use the variables to its left (shared), so we can begin by looking for definitions of \(x_i\) in terms of shared. Towards constructing the formula for the corresponding unsatisfiability check according to Theorem 2, COPY(\(\varphi, X\)) returns a copy \(\varphi'\) of the matrix \(\varphi\) where each variable \(x \in \text{var}(\varphi) \setminus \text{shared}\) has been replaced by a fresh variable \(x'\). Next (lines 9–14), we consider each variable \(x_j\) with quantifier type \(Q\)—these are the variables we want to find definitions of—and introduce two fresh “selector” variables \(s_i\) and \(s'_i\), while adding clauses \((\neg s_j \lor x_j)\) and \((\neg s'_j \lor \neg x'_j)\) to \(\varphi\) and \(\varphi'\), respectively. These clauses allow us to represent \(\varphi \land x_j \land \varphi' \land \neg x'_j\) by assuming literals \(s_j\) and \(s'_j\).

After initializing the SAT solver, we consider the variables \(x_1, \ldots, x_n\) in the order of the quantifier prefix (lines 18–29). If variable \(x_j\) has quantifier type \(Q\), we want to check whether \(x_j\) is defined in \(\varphi\) in terms of oppositely quantified variables \(X_j\) that precede it in the prefix (Proposition 1 tells us that in this case the strategy function of \(x_j\) is unique). For the first such variable \(x_j\), it is clear that the set of variables common to \(\varphi\) and \(\varphi'\) is precisely \(X\). Unsatisfiability of \(\varphi \land x_j \land \varphi' \land \neg x'_j\) is decided by calling the SAT solver under assumptions \(\{s_j, s'_j\}\): the assumptions ensure that \(x_j\) and \(\neg x'_j\) are set to true by propagation, and all remaining selector variables can be set to false so as to satisfy the clauses they occur in without interfering with the remaining clauses. If the solver determines unsatisfiability, an interpolant \(I_j\) is computed (line 22), which by Theorem 2 corresponds to a definition of \(x_j\), and adds the pair \((x_j, I_j)\) to a list of definitions. Otherwise, if \(x_j\) has the quantifier type opposite to \(Q\), the strategy function of any variable with quantifier type \(Q\) considered later may use \(x_j\). Accordingly (lines 26–27), we add clauses \((x_j \lor \neg x'_j)\) and \((\neg x_j \lor x'_j)\) to \(\varphi'\) through the incremental interface of the SAT solver. This has two effects: first, it enforces equivalence of \(x_j\) and \(x'_j\), and second, \(x_j\) is added to the common vocabulary of \(\varphi\) and \(\varphi'\), so that it can appear in interpolants computed in later iterations.

Soundness of Algorithm 1 as stated in the following proposition can be proved by a straightforward induction on the quantifier prefix using Theorem 2 and Proposition 1.

**Proposition 2.** Given a quantified Boolean formula \(\Phi\) and a quantifier type \(Q \in \{\forall, \exists\}\), Algorithm 1 terminates with a (possibly empty) set \(\{ (x_1, I_1), \ldots, (x_k, I_k) \}\) of pairs \((x_i, I_i)\) such that \(I_i\) represents the unique strategy function of \(x_i\) in \(\Phi\) and \(\text{var}(x_i) \in \text{var}_Q(\Phi)\) for \(1 \leq i \leq k\).

**Example 2.** Consider the QBF \(\Psi = \forall x_1 \exists y_1 \forall x_2 \exists y_2 \varphi\), where

\[
\varphi = (x_1 \lor y_1) \land (\neg x_1 \lor \neg y_1) \land (x_2 \lor y_2) \land (\neg x_2 \lor \neg y_2).
\]

---

4 Two distinct selector variables are required to ensure that they do not belong to the common variables of \(\varphi\) and \(\varphi'\).

5 One could also add these clauses to \(\varphi\), in which case \(x'_j\) would become part of the shared vocabulary. This has the slight disadvantage that subsequently computed definitions may use a mixture of variables from \(\varphi\) and \(\varphi'\), rather than just \(\varphi\).
Algorithm 1. Extracting Unique Strategy Functions by Interpolation

1: procedure getDefinitionsQBF($\Phi, Q \in \{\forall, \exists\}$)
2:     $Q_1 \ldots Q_n x_n. \varphi \leftarrow \Phi$
3:     $i = \min\{1 \leq i \leq n \mid Q_i = Q\}$
4:     $\text{shared} \leftarrow \{x_1, \ldots, x_{i-1}\}$
5:     if $Q = \forall$ then
6:         $\varphi \leftarrow \neg \varphi$ \quad $\triangleright \forall$-strategies aim to falsify the matrix.
7:     end if
8:     $\varphi' \leftarrow \text{COPY}(\varphi, \text{shared})$
9:     $\text{sametype} \leftarrow \{j \mid 1 \leq j \leq n \text{ and } Q_j = Q\}$
10:    for $j \in \text{sametype}$ do
11:        $s_j, s'_j \leftarrow$ fresh variables
12:        $\varphi \leftarrow \varphi \land (\neg s_j \lor x_j)$
13:        $\varphi' \leftarrow \varphi' \land (\neg s'_j \lor \neg x'_j)$
14:    end for
15:    $\text{solver} \leftarrow \text{ItpSatSolver}(\varphi, \varphi')$
16:    $\text{defined} \leftarrow \emptyset$
17:    $k \leftarrow \max\{i \leq k \leq n \mid Q_k = Q\}$
18:    for $j = i, \ldots, k$ do
19:        if $Q_j = Q$ then
20:            result $\leftarrow \text{solver.solve}\{s_j, s'_j\}$
21:            if result = UNSAT then
22:                $I_j \leftarrow \text{solver.getInterpolant()}$
23:                $\text{defined} \leftarrow \text{defined} \cup \{(x_j, I_j)\}$
24:            end if
25:        else
26:            $\text{solver.addClause}(\varphi', x_j \lor \neg x'_j)$
27:            $\text{solver.addClause}(\varphi', \neg x_j \lor x'_j)$
28:        end if
29:    end for
30:    return defined
31: end procedure

We illustrate a run of Algorithm 1 on $\Psi$ with $Q = \exists$. Since $y_1$ is the leftmost existential variable, we create a copy $\varphi'$ of $\varphi$ with every variable except $x_1$ renamed, that is,

$$\varphi' = (x_1 \lor y'_1) \land (\neg x_1 \lor \neg y'_1) \land (x'_2 \lor y'_2) \land (\neg x'_2 \lor \neg y'_2).$$

We also add the clauses $(\neg s_1 \lor y_1)$ and $(\neg s_2 \lor y_2)$ to $\varphi$ and the clauses $(\neg s'_1 \lor \neg y'_1)$ and $(\neg s'_2 \lor \neg y'_2)$ to $\varphi'$. In the main loop, Algorithm 1 first checks whether $\varphi \land \varphi'$ is unsatisfiable under the assumptions $\{s_1, s'_1\}$. Unit propagation simplifies $\varphi$ to (omitting unused selector variables and clauses)

$$(\neg x_1) \land (x_2 \lor y_2) \land (\neg x_2 \lor \neg y_2),$$

and $\varphi'$ simplifies to

$$(x_1) \land (\neg x'_2 \lor y'_2) \land (\neg x'_2 \lor \neg y'_2).$$
By resolving \((\neg x_1)\) with \((x_1)\) we obtain the empty clause, and \(\neg x_1\) is the corresponding interpolant,\(^6\) so \((y_1, \neg x_1)\) is added to the set of definitions. Next, we consider the universally quantified variable \(x_2\) and add the clauses \((x_2 \lor \neg x_2')\) and \((\neg x_2 \lor x_2')\) to \(\varphi'\). Finally, we check whether \(y_2\) is definable by calling the SAT solver under the assumptions \(\{s_2, s_2'\}\). Now, the formula \(\varphi\) simplifies to \((x_1 \lor y_1) \land (\neg x_1 \lor \neg y_1) \land (\neg x_2)\), and \(\varphi'\) simplifies to \((x_1' \lor y_1') \land (\neg x_1' \lor \neg y_1') \land (x_2') \land (x_2 \lor \neg x_2') \land (\neg x_2 \lor x_2')\).

Unit propagation derives the clause \((x_2)\) from the clauses in the second line, which can be resolved with the clause \((\neg x_2)\) from \(\varphi\) to obtain a resolution refutation of the formula \(\varphi \land \varphi'\), with \(\neg x_2\) as an interpolant. Accordingly, \((y_2, \neg x_2)\) is added to the set of definitions. Algorithm 1 terminates with the definitions \(\{(y_1, \neg x_1), (y_2, \neg x_2)\}\), and it is readily verified that \(y_1 \equiv \neg x_1\), \(y_2 \equiv \neg x_2\) is indeed the unique existential winning strategy of \(\Psi\).

### 4.2 Improvements and Generalization to Dependency QBF

Consider a QBF \(\Phi = \forall x_1, x_2 \exists y_1, y_2 . (x_1 \leftrightarrow x_2) \leftrightarrow (y_1 \leftrightarrow y_2)\). It is easy to verify that \(\Phi\) is true and that \(y_1\) and \(y_2\) do not have unique Skolem functions: for every assignment to the universal variables there are two ways of setting \(y_1\) and \(y_2\) so as to satisfy the matrix, so neither existential variable is defined by the universal variables alone. However, each variable is defined by all remaining variables. For instance, variable \(y_2\) is defined by \(x_1, x_2,\) and \(y_1\).

More generally, increasing the set of defining variables allows us to detect more definitions: if \(x\) is defined in terms of \(X\) then it is also defined in terms of any enclosing set \(X' \supset X\). To exploit this, we modified Algorithm 1 so as to assume a total ordering of variables and check for definitions of a variable \(x\) in terms of all variables \(X\) which precede it in the quantifier prefix. This can be implemented by simply adding clauses encoding equivalence of \(x_j\) and \(x_j'\) (lines 26–27) regardless of quantifier type.

Technically, this leads to an alternative definition of a “winning strategy” for a QBF where each strategy function takes an assignment to all preceding variables as input. Both definitions are ultimately equivalent in the sense that a winning strategy according to one definition can be transformed into a winning strategy according to the other definition without changing its responses (cf. the work on quantifier elimination by functional composition and self-substitution [8,14,28,29]). One can prove an analogue of Proposition 1 stating that the strategy function—according to the alternative definition—of a variable \(x\) is unique whenever \(x\) is defined in terms of the variables preceding \(x\) in the quantifier prefix.

\(^6\) As mentioned above, interpolants can be efficiently extracted from resolution refutations [32,36,46].
Dependency Quantified Boolean Formulas (DQBFs) generalize QBFs by allowing a non-linear quantifier prefix. More specifically, each existential variable is annotated with a set of universal variables its Skolem function may depend on. A DQBF is true if there is an existential winning strategy such that each Skolem function satisfies these restrictions [2]. Although evaluating DQBF is NEXPTIME-complete and thus believed to be much harder than evaluating QBF, the fact that problems can be concisely encoded in DQBF [12,18] has prompted the development of dedicated DQBF solvers [13,15,17,48].

Algorithm 1 can easily be extended to compute unique Skolem functions of DQBF. The standard DQDIMACS format [15] allows for the combination of a linear quantifier prefix with variables for which the dependency sets are explicitly stated. The linear quantifier prefix can be handled as before. For each existential variable \(x\) with explicit dependency set \(D_x\) we simply check whether \(x\) is defined by \(D_x\). If multiple variables \(x_1, \ldots, x_k\) have the same dependency set \(D_x\) (which is frequently the case in benchmark formulas) we check whether \(x_i\) is defined by \(D_x \cup \{x_1, \ldots, x_{i-1}\}\) for each \(1 \leq i \leq k\). Again, this technically requires a non-standard definition of Skolem functions for DQBF but can easily be proven sound.

5 Implementation

We implemented the algorithm described in the previous section in a prototype named UNIQUE. As a back end SAT solver we use ItpMiniSAT, a modified version of MiniSAT [11] bundled with the EXTAVY model checker that efficiently generates interpolants in memory and supports both assumptions and incremental solving [22,49]. UNIQUE can read PCNF formulas (QDIMACS), prenex non-CNF QBFs (QCIR), as well as DQBFs with CNF matrices (DQDIMACS).

Interpolants obtained from ItpMiniSAT are represented as And-Inverter graphs (AIGs) and accessed through the AIG library of ABC [7]. To make use of the structural sharing capabilities of AIGs, we maintain a single AIG representing the interpolants computed in the main loop (lines 18–29) of Algorithm 1. Whenever a new interpolant is obtained, the corresponding AIG returned by ItpMiniSAT is merged into the existing AIG. If the number of AIG nodes exceeds a (geometrically increasing) threshold, we use the ABC macro compress2 to reduce the size of the combined AIG. Upon termination, and assuming the AIG is not too large, this is followed up by a round of FRAIGing [37] and a final application of compress2.

While running UNIQUE on QBFs with multiple quantifier alternations we noticed that ItpMiniSAT got stuck attempting to solve some of the definability queries. Further testing revealed that the corresponding instances were hard for most state-of-the-art solvers. Increasing the overall timeout would allow us to solve these instances in some cases, but naturally the corresponding interpolants (for unsatisfiable instances) were very large (and difficult to compress with ABC). This clearly defeats the purpose of detecting unique strategy functions quickly. We thus decided to impose a limit on the number of conflicts.
for each call of ITPI MINISAT (currently set to 1000 conflicts). This significantly reduces the overall running time of UNIQUE for many instances and ensures that individual interpolants are small, but only marginally decreases the total number of definitions found.

Since the individual definability queries are independent of each other, it is not necessary to determine for each input variable whether it is defined. Accordingly, we implemented UNIQUE as an anytime algorithm: upon termination, it returns the set of variables with unique strategy functions identified up to that point, along with the AIG representing the corresponding functions.

6 Experiments

For the experiments described below we used a cluster with Intel Xeon E5649 processors at 2.53 GHz running 64-bit Linux.

6.1 Gate Extraction

We first ran UNIQUE to compute unique strategy functions for the instances in the 2QBF (402 instances) benchmark set from the 2018 QBF Evaluation, as well as the PCNF (558), QCIR (341), and DQBF (333) benchmark sets from the 2019 QBF Evaluation. For each job we imposed a time limit of 600 s and a memory limit of 1.8 GB.

![Figure 1](http://www.qbflib.org)

**Fig. 1.** Running time (s) of UNIQUE by benchmark set. For each 50-s interval within the time limit (x-axis), the number of instances (y-axis) processed by UNIQUE with a running time in that interval is shown.

Figure 1 shows a histogram for the running time of UNIQUE on different benchmark sets. While most instances are processed quickly, UNIQUE runs into
the time limit for a significant number of PCNF instances. Generally, the running time increases with the size of the matrix and the number of variables. This explains why almost all DQBF formulas are processed quickly, as these tend to be much smaller compared to formulas from the other benchmark sets.

Figure 2 shows a histogram for the fraction of existential variables with unique strategy functions in 2QBF and PCNF instances (turquoise bars). We clearly see a bimodal distribution here: there is a large number of instances where the strategy functions of most variables are unique, but also a significant number of instances where few existential strategy functions are unique. To determine how many of the corresponding definitions cannot be found by syntactic gate detection, we used the QCIR-conv script provided by GhostQ [31] to convert 2QBF and PCNF instances to QCIR, and ran UNIQUE again on the resulting circuits. To do this, the circuit is translated (back) to CNF, but auxiliary variables representing gates are ignored by the definability check. Testing showed that a one-sided CNF encoding [42] works better than standard Tseitin conversion.

![Figure 2](image)

**Fig. 2.** Fraction of existential variables with unique strategy functions in 2QBF (left) and PCNF (right) instances before (turquoise) and after (red) syntactic gate detection. For each fraction (x-axis) we see the number of instances (y-axis) with the corresponding fraction of unique existential strategy functions. (Color figure online)

Table 1 (left) shows quartiles for the distributions of unique existential strategy functions detected by UNIQUE in each benchmark set. We only show the distribution for existential variables in Table 1 and Fig. 2 since very few universal variables were found to have unique strategy functions. In fact, only 51 instances from the QCIR benchmark set encoding bounded synthesis for Petri games contained such universal variables.

The fraction of variables with unique strategy functions was smallest for QCIR instances. This is expected, since they can represent circuit structure directly and do not require auxiliary variables to encode gate definitions. By

---

8 For instance, the left side of the first row of Table 1 says that for 75% of 2QBF instances, UNIQUE was able to identify 3% of Skolem functions as unique; for half of the instances, at least 90% of existential variables were identified as having unique Skolem functions; and for 25% of instances, at least 96%.
Table 1. Distribution (quartiles) of the fraction of unique Skolem functions identified by UNIQUE before (left) and after (right) preprocessing with HQSPre. Rows marked by a star (*) show the distribution after syntactic gate detection.

|       | Original |       |       | Preprocessed |       |       |
|-------|----------|-------|-------|--------------|-------|-------|
|       | 1st      | Median| 3rd   | 1st          | Median| 3rd   |
| 2QBF  | 0.03     | 0.9   | 0.96  | 0            | 0     | 0     |
| 2QBF* | 0        | 0.22  | 0.54  | 0            | 0     | 0     |
| PCNF  | 0        | 0.53  | 0.94  | 0            | 0     | 0.03  |
| PCNF* | 0        | 0.21  | 0.53  | 0            | 0     | 0.02  |
| QCIR  | 0        | 0     | 0.13  | –            | –     | –     |
| DQBF  | 0.57     | 0.88  | 0.94  | 0.22         | 0.45  |       |

contrast, 2QBF and DQBF instances contain many variables with unique strategy functions. For about half of the instances, between roughly 90% and 95% of the existential strategy functions are unique.

On the right of Table 1 we show the distribution of unique existential strategy functions after preprocessing with HQSPre [52]. Clearly, only very few unique Skolem functions are detected by UNIQUE. This may be in part due to the fact that preprocessing detects and removes gate definitions [27]. Another possibility is that definitions are simply lost: some of the most powerful preprocessing techniques for QBF currently used only preserve the truth value and not the set of strategies [23]. We will return to this topic at the end of the next subsection.

6.2 Solving Formulas Augmented with Definitions

Unique strategy functions of a (D)QBF can be substituted for their variables without changing the set of winning strategies. This can be used in preprocessing to reduce the number of quantified variables, typically at the cost of increasing the size of the matrix. In the following experiments, we substituted definitions found by UNIQUE for the defined variables and ran QBF and DQBF solvers on the resulting instances.

First, we considered the 2QBF benchmark set. We picked the QCIR solvers QUABS [47], QFUN [26], and GHOSTQ [31], along with the dedicated 2QBF (PCNF) solver CADET [43]. For the QCIR solvers, the performance on instances constructed by syntactic gate detection with QCIR-conv serves as a baseline. We compare it with performance on instances obtained by UNIQUE and—since QCIR-conv also performs circuit-level simplifications that go beyond gate extraction—with a combination of both where QCIR-conv and UNIQUE are run in sequence.

For CADET, we compare performance on the original 2QBF instances with performance on QDIMACS instances augmented with CNF encodings of definitions extracted by UNIQUE. For each configuration, we report the number of
instances solved within a time limit of 15 min. To isolate the effect of adding definitions, the time required by UNIQUE (and QCIR-Conv) is not counted towards the time limit. The results are shown in Fig. 3 (left).

![Gate Detection](image)

**Fig. 3.** Number of 2QBF instances solved (y-axis) by solvers (x-axis) using different gate detection methods before (left) and after (right) preprocessing with HQSPRe.

QFUN, QUABS, and GHOSTQ benefit considerably from semantic gate extraction, in particular when applied on top of syntactic gate extraction. By contrast, CADET solves fewer instances augmented with gate definitions than original instances. We found this surprising, since variable definitions should be detected by CADET’s heuristic for identifying unique Skolem functions. Perhaps most definitions found by UNIQUE are already covered in this way, so that the additional clauses simply slow down propagation. We believe that explicitly telling CADET which variables have already been identified as determined should result in a speedup overall.

Figure 4 takes a closer look at solving times for individual instances (for this plot, memory outs are treated as timeouts). CADET is slower on instances augmented by UNIQUE but fairly consistent, while the effect on the other solvers is more erratic. We conjecture that this is because the set of existential strategies is preserved and the instances thus “look similar” to CADET.

Next, we tested with PCNF instances and considered the QDIMACS solvers DEPQBF [5] and CAQE [44], as well as the QCIR solvers QUABS [47], QFUN [26], and QUTE [40]. Again, we compare the number of instances solved in 15 min with different options for gate detection. Results are shown in Fig. 5 (left). Again all QCIR solvers benefit from gate detection with UNIQUE when performed on top of syntactic gate detection with QCIR-Conv, while performance

---

9 The results are qualitatively the same when the running time of UNIQUE is counted towards the time limit: the largest decrease in the number of solved instances across all benchmark sets and configurations is 7.
Fig. 4. Solving time (s) for 2QBF instances with (x-axis) and without \textsc{Unique} (y-axis).

decreases for both QDIMACS solvers. The additional clauses and variables introduced by \textsc{Unique} apparently do not help these solvers and simply result in a slowdown.

Finally, we tested the impact of \textsc{Unique} on DQBF (DQDIMACS) instances solved by HQS [19] and DCAQE [48] within 15 min. Since DQBF solvers currently do not (yet) support non-CNF input, we translate definitions to CNF and add them to the original formulas. Note that whenever an existential variable $x$ is defined by (a subset of) its dependency set, we can safely let $x$ depend on additional variables. This is sound since the response of variable $x$ is already determined by the variables in the original dependency set and cannot change depending on other inputs. In particular, we can collect all defined variables (and auxiliary variables) in an “innermost” existential quantifier block that depends on all universal variables. Since many existential variables have uniquely determined strategy functions (see Table 1), this allows us to push many variables into the innermost quantifier block and get closer to a linear quantifier prefix. For HQS, this translates into a small increase in the number of solved instances (208 vs. 189), whereas DCAQE basically solves the same number of instances (133 vs. 135).

\textit{Interaction with Preprocessing.} QBF solvers for PCNF are typically paired with preprocessors such as BLoQQEr [6] or HQSPre [52]. These are highly engineered tools that batter instances with a barrage of techniques and can often solve formulas completely on their own. Most solvers benefit greatly from
preprocessing. This is evident in Fig. 5 (right), which shows the number of solved PCNF instances with different forms of gate detection after preprocessing with HQSPre (within a timeout of 600 s). Here, the number of solved instances increases significantly for almost all systems.

At the same time, preprocessing appears to obscure or destroy definitions. UNIQUE hardly finds any definitions in preprocessed instances (cf. Table 1) and accordingly has little impact on performance. For QFUN, which benefitted most from gate detection in our experiments, this translates to a substantial reduction in the number of solved instances. On the 2QBF benchmark set (Fig. 3), both QFUN and GHOSTQ solve significantly fewer instances with HQSPRE compared to the combination of UNIQUE and QCIR-Conv, whereas the number of solved instances almost doubles for QuAbS. Understanding which preprocessing techniques obscure gate definitions and why certain solvers benefit more from gate detection than others are important questions for future work.¹⁰

7 Related Work

Our semantic gate detection technique is closely related to a method for determinizing Boolean relations by Jiang et al. [29], a problem that essentially corresponds to solving 2QBF. The authors show that, for a (total) relation $R(X, y)$ with a single output variable $y$, a functional implementation of $y$ can be obtained as an interpolant for $\neg R(X, 0) \land \neg R(X, 1)$. This can be used to determinize

¹⁰ We also ran experiments with QCIR-Conv and UNIQUE applied before preprocessing. The results were significantly worse, so we do not report them in detail. Standard preprocessing requires PCNF input, so that definitions have to be encoded using additional clauses and Tseitin variables. Just like the PCNF solvers in the other experiments, HQSPRE appears to be unable to do anything useful with these extra clauses and variables.
relations \( R(X,Y) \) with a set of output variables \( Y = \{ y_1, \ldots, y_n \} \). First, an implementation \( f_n \) for \( y_n \) can be computed by treating \( R \) as a relation with inputs \( X \cup \{ y_1, \ldots, y_{n-1} \} \) and single output \( y_n \). Subsequently, the implementation \( f_n \) can be substituted for \( y_n \) to obtain a relation \( R'(X,Y \setminus \{ y_n \}) \). By repeating this process, a functional implementation \( f_1 \) of \( y_1 \) can eventually be obtained. Substituting \( f_i \) into \( f_{i+1} \) for \( 1 \leq i < n \) results in functional implementations that only depend on the original input variables \( X \). This approach does not require for any of the output variables to be defined by \( X \), but an implementation of \( y_i \) solely in terms of the input variables \( X \) is only available at the very end of this process. For deterministic relations \( R(X,Y) \) (where every \( y \) is defined in terms of \( X \)), the authors show that a functional implementation of \( y \in Y \) can be obtained as the interpolant of a formula that corresponds to the formula in the statement of Padoa’s theorem. Our result stated as Theorem 2 is more general in that it holds for multi-output relations that are not necessarily deterministic.

Hofferek et al. use interpolation to synthesize multiple functional implementations from a single proof and thus avoid the increase in formula size incurred by repeated substitution [24]. This has an analogue in strategy extraction for QBF, which allows for implementations of all (existential or universal) variables to be obtained from a proof [3]. However, strategy extraction requires the input QBF has been solved, whereas our main interest is in preprocessing QBF.

There is a series of works on recovering gate definitions from CNF formulas. Li integrated rules for detecting equivalent literals in a Davis-Putnam style algorithm [35]. Ostrowski et al. represent formulas as graphs to detect patterns corresponding to and-gates, or-gates, and equivalences [38]. Roy et al. use CNF signatures to detect a richer set of gates [45]. Fu and Malik extend this to arbitrary (user-specified) gate libraries and ensure that a maximum acyclic circuit is constructed [16].

In the context of QBF, Bacchus and Goultiaeva showed that circuit reconstruction can speed up solvers by providing them with a better set of initial cubes [21]. They also extended the scope of these techniques to CNF formulas obtained from circuits by the Plaisted-Greenbaum encoding [42]. Scholl and Pigorsch developed a QBF solver that manipulates an AIG representation of the matrix to perform quantifier elimination and relies on circuit reconstruction to simplify the initial AIG [41].

Balabanov et al. proposed a SAT-based semantic gate extraction technique [4]. Their approach has the disadvantage that a subset of clauses inducing a definition has to be guessed. As a more efficient heuristic, they suggest to identify pseudo definitions instead. A set of clauses \( (A_1 \lor x), \ldots, (A_k \lor x), (B_1 \lor \neg x), \ldots, (B_l \lor \neg x) \) is a pseudo definition of \( x \) if the formula \( A_1 \land \cdots \land A_k \land B_1 \land \cdots \land B_l \) is unsatisfiable. Rabe and Seshia use a similar criterion in their incremental determinization algorithm to identify variables that are (locally) deterministic [43]. Checking for pseudo definitions is typically efficient but limits the range of definitions that can be detected.
8 Conclusion

Syntactic gate detection has been shown to benefit SAT solvers [10, 16] and QBF solvers [21]. The underlying algorithms are fast but limited to a predefined library of gates. By contrast, our semantic gate extraction method can detect any definition entailed by an input formula but requires an interpolating SAT solver. In the context of SAT, this overhead likely outweighs any potential benefits. However—as demonstrated by our experiments—there is significant potential for application to harder problems such as QBF and DQBF evaluation. Here, preprocessing is just a first step.

At the same time, our results show that substituting unique strategy functions can slow down solvers. In some sense, this is counter-intuitive: ideally, providing solvers with unique strategy functions should give them a head start, or at least not hurt their performance. By analogy, if we give a SAT solver part of a backbone assignment, it can simply instantiate accordingly and need not consider the corresponding variables for the remainder of its run. With the exception of CADET, QBF solvers currently cannot “instantiate” variables with strategy functions in this way, since they are only equipped to reason about assignments. We believe that designing techniques for reasoning about strategies is a key challenge in developing the next generation of QBF solvers.

Acknowledgements. The author would like to thank Adrian Rebola-Pardo, Matthias Schlaipfer, and Georg Weissenbacher for helpful discussions.

References

1. Ansótegui, C., Gomes, C.P., Selman, B.: The Achilles’ heel of QBF. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI 2005, pp. 275–281. AAAI Press/The MIT Press (2005)
2. Balabanov, V., Chiang, H.J.K., Jiang, J.R.: Henkin quantifiers and Boolean formulae: a certification perspective of DQBF. Theor. Comput. Sci. 523, 86–100 (2014)
3. Balabanov, V., Jiang, J.R.: Unified QBF certification and its applications. Formal Methods Syst. Des. 41(1), 45–65 (2012)
4. Balabanov, V., Jiang, J.R., Mishchenko, A., Scholl, C.: Clauses versus gates in CEGAR-based 2QBF solving. In: Darwiche, A. (ed.) Beyond NP, Papers from the 2016 AAAI Workshop, AAAI Workshops, vol. WS-16-05. AAAI Press (2016)
5. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 158–171. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14186-7_14
6. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 101–115. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22438-6_10
7. Brayton, R., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 24–40. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_5
8. Bubeck, U., Kleine Büning, H.: Nested boolean functions as models for quantified boolean formulas. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 267–275. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39071-5_20

9. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. Symb. Log. 22(3), 269–285 (1957)

10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005). https://doi.org/10.1007/11499107_5

11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_20

12. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of bounded synthesis. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 354–370. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_20

13. Finkbeiner, B., Tentrup, L.: Fast DQBF refutation. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 243–251. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_19

14. Fried, D., Tabajara, L.M., Vardi, M.Y.: BDD-based boolean functional synthesis. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 402–421. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_22

15. Fröhlich, A., Kovácsznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Pragmatics of SAT 2014 (2014)

16. Fu, Z., Malik, S.: Extracting logic circuit structure from conjunctive normal form descriptions. In: VLSI Design 2007, (ICES 2007), pp. 37–42. IEEE Computer Society (2007)

17. Ge-Ernst, A., Scholl, C., Wimmer, R.: Localizing quantifiers for DQBF. In: Barrett, C.W., Yang, J. (eds.) 2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, 22–25 October 2019, pp. 184–192. IEEE (2019)

18. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulæ. In: 31st International Conference on Computer Design, ICCD 2013, pp. 396–403. IEEE (2013)

19. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: Nebel, W., Atienza, D. (eds.) DATE 2015, pp. 1617–1622. ACM (2015)

20. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 89–134. Elsevier (2008)

21. Gouliaeva, A., Bacchus, F.: Recovering and utilizing partial duality in QBF. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 83–99. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39071-5_8

22. Gurfinkel, A., Vizel, Y.: Druping for interpolates. In: FMCAD 2014, pp. 99–106. IEEE (2014)

23. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. J. Artif. Intell. Res. 53, 127–168 (2015)

24. Hofferek, G., Gupta, A., Köninghofer, B., Jiang, J.R., Bloem, R.: Synthesizing multiple Boolean functions using interpolation on a single proof. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, 20–23 October 2013, pp. 77–84. IEEE (2013)
25. Ivrii, A., Malik, S., Meel, K.S., Vardi, M.Y.: On computing minimal independent support and its applications to sampling and counting. Constraints 21(1), 41–58 (2016)
26. Janota, M.: Towards generalization in QBF solving via machine learning. In: McIlraith, S.A., Weinberger, K.Q. (eds.) AAAI-18, pp. 6607–6614. AAAI Press (2018)
27. Järvisalo, M., Biere, A., Heule, M.: Simulating circuit-level simplifications on CNF. J. Autom. Reasoning 49(4), 583–619 (2012)
28. Jiang, J.-H.R.: Quantifier elimination via functional composition. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 383–397. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_30
29. Jiang, J.R., Lin, H., Hung, W.: Interpolating functions from large Boolean relations. In: Roychowdhury, J.S. (ed.) ICCAD 2009, pp. 779–784. ACM (2009)
30. Jordan, C., Klieber, W., Seidl, M.: Non-CNF QBF solving with QCIR. In: Darwiche, A. (ed.) Beyond NP, Papers from the 2016 AAAI Workshop. AAAI Workshops, vol. WS-16-05. AAAI Press (2016)
31. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A non-prenex, non-clausal QBF solver with game-state learning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 128–142. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14186-7_12
32. Krajícek, J.: Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. J. Symb. Log. 62(2), 457–486 (1997)
33. Lagniez, J., Lonca, E., Marquis, P.: Improving model counting by leveraging definability. In: Kambhampati, S. (ed.) IJCAI 2016, pp. 751–757. IJCAI/AAAI Press (2016)
34. Lang, J., Marquis, P.: On propositional definability. Artif. Intell. 172(8–9), 991–1017 (2008)
35. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA, pp. 291–296. AAAI Press/The MIT Press (2000)
36. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_1
37. Mishchenko, A., Chatterjee, S., Brayton, R.: FRAIGs: A unifying representation for logic synthesis and verification. Technical report, Berkeley (2005)
38. Ostrowski, R., Grégoire, É., Mazure, B., Säis, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 185–199. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3_13
39. Padoa, A.: Essai d’une théorie algébrique des nombres entiers, précédé d’une Introduction logique à une théorie déductive quelconque. Bibliothèque du Congrès International de Philosophie (1903)
40. Peitl, T., Slivovsky, F., Szeider, S.: Dependency learning for QBF. J. Artif. Intell. Res. 65, 180–208 (2019)
41. Pigorsch, F., Scholl, C.: Exploiting structure in an AIG based QBF solver. In: Benini, L., Micheli, G.D., Al-Hashimi, B.M., Müller, W. (eds.) DATE 2009, pp. 1596–1601. IEEE (2009)
42. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. 2(3), 293–304 (1986)
43. Rabe, M.N., Seshia, S.A.: Incremental determinization. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 375–392. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_23

44. Rabe, M.N., Tentrup, L.: CAQE: A certifying QBF solver. In: Kaivola, R., Wahl, T. (eds.) FMCAD 2015, pp. 136–143. IEEE Computer Soc. (2015)

45. Roy, J., Markov, I., Bertacco, V.: Restoring circuit structure from SAT instances. In: Proceedings of International Workshop on Logic and Synthesis, pp. 663–678 (2004)

46. Schlaipfer, M., Weissenbacher, G.: Labelled interpolation systems for hyper-resolution, clausal, and local proofs. J. Autom. Reasoning 57(1), 3–36 (2016)

47. Tentrup, L.: Non-prenex QBF solving using abstraction. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 393–401. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_24

48. Tentrup, L., Rabe, M.N.: Clausal abstraction for DQBF. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 388–405. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_27

49. Vizel, Y., Gurfinkel, A., Malik, S.: Fast interpolating BMC. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 641–657. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_43

50. Vizel, Y., Weissenbacher, G., Malik, S.: Boolean satisfiability solvers and their applications in model checking. Proc. IEEE 103(11), 2021–2035 (2015)

51. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31

52. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre – an effective preprocessor for QBF and DQBF. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 373–390. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_21

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.