Solving QBF by Abstraction*

Jesko Hecking-Harbusch
Reactive Systems Group
Saarland University

Leander Tentrup
Reactive Systems Group
Saarland University

Many verification and synthesis approaches rely on solving techniques for quantified Boolean formulas (QBF). Consequently, solution witnesses, in the form of Boolean functions, become more and more important as they represent implementations or counterexamples. We present a recursive counterexample guided abstraction and refinement algorithm (CEGAR) for solving and certifying QBFs that exploits structural reasoning on the formula level. The algorithm decomposes the given QBF into one propositional formula for every block of quantifiers that abstracts from assignments of variables not bound by this quantifier block. Further, we show how to derive an efficient certification extraction method on top of the algorithm. We report on experimental evaluation of this algorithm in the solver QuAbS (Quantified Abstraction Solver) which won the most recent QBF competition (QBFEVAL'18). Further, we show the effectiveness of the certification approach using synthesis benchmarks and a case study for synthesizing winning strategies in Petri Games.

1 Introduction

Synthesis is the task to produce correct-by-design implementations from formal specifications. This allows the developer to focus on what to achieve in form of the specification instead of focussing on how to implement requirements. The synthesis task is usually formulated as a two-player game between the *system* player whose objective is to satisfy the specification and the *environment* player who tries to falsify the specification. There are many variants of such games in literature, suitable for different types of systems, such as synchronous, asynchronous, and distributed ones, and for different kinds of objectives, such as safety objectives, ω-regular winning conditions, and beyond. Determining the winner of a synthesis game, which is equivalent to the answer whether the underlying specification is *realizable*, gives us the knowledge of whether or not an implementation exists that satisfies the specification. In the best case, we can directly construct an implementation from a winning strategy of the synthesis game.

In this paper, we consider the satisfiability problem of quantified Boolean formulas (QBF), which can be formulated as a game between the *existential* and *universal* player, controlling the existential and universal quantifiers, respectively. QBF has been used to encode the realizability problem for many of the specifications and games mentioned before, such as symbolically represented safety games [3], the LTL realizability problem [5], distributed and fault-tolerant synthesis [11,12], and asynchronous systems using Petri games [7]. As a side-effect of those encodings, a certification of the QBF solving result in many cases directly corresponds to winning strategies and implementations. *QBF certification* is the task to extract *Skolem* functions for the existential quantifiers of true QBFs and *Herbrand* functions for the universal quantifiers of false QBFs.

Despite its benefits, QBF certification is a weak spot of current solving algorithms. There are a number of works in the literature [1, 16, 27, 32] for certifying QBFs given in conjunctive normal form (CNF), but in practice it involves performance penalties due to non-applicable solving optimizations and

*Supported by the German Research Foundation (DFG) Grant Petri Games (No. 392735815) and by the European Research Council (ERC) Grant OSARES (No. 683300).

Submitted to:
GandALF 2018

limited preprocessing[1]. We believe that certification must be treated as a first-class citizen of QBF solving and show that it is possible to have competitive performance and solution extraction at the same time. A crucial approach to this goal is that we consider formulas in negation normal form (NNF) instead of CNF. Using (the less restrictive) NNF, the QBF solving problem becomes dual with respect to negation, removing the inherent imbalance of CNF solving algorithms [24].

We present a counterexample guided abstraction and refinement (CEGAR) algorithm for solving quantified Boolean formulas that exploits the propositional formula's structure. The algorithm decomposes the given QBF into one propositional formula for every maximal block of consecutive quantifiers of the same type. We call this formula an abstraction, because it *abstracts* from assignments of variables that are not bound by this block of quantifiers of the same type. We use special *interface variables* to communicate *assumptions* (outer-to-inner quantifier) and *learned information* (inner-to-outer quantifier) during solving. Further, we use a SAT solver as an oracle to generate new abstraction entries and to provide us with witnesses for unsatisfiable queries. Given a QBF, the algorithm proceeds by generating a candidate solution using a SAT solver and the abstraction. Then, this candidate is verified (or refuted) recursively and, depending on the result, the abstraction is refined. We introduce a new element to QBF refinement algorithms by maintaining, for every quantifier block, a *dual abstraction* and make twofold use of it: It provides a method for optimizing abstraction entries and it is used to translate counterexamples from one quantifier block to another.

To sum up, this paper makes the following contributions:

- We provide a counterexample guided abstraction and refinement (CEGAR) algorithm for solving QBFs in negation normal form.
- We describe an efficient certification approach and evaluate it on synthesis benchmarks where implementations can be obtained from certificates.
- As a case study, we show how to make use of the certification feature to build strategies representing implementations and counterexamples for Petri games.

2 Quantified Boolean Formulas

A quantified Boolean formula (QBF) is a propositional formula over a finite set of variables $\mathscr{X}$ extended with quantification. The syntax is given by the grammar

$$\varphi \coloneqq x \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi \quad ,$$

where $x \in \mathscr{X}$. For readability, we lift the quantification over variables to the quantification over sets of variables and denote a maximal consecutive block of quantifiers of the same type $\forall x_1.\, \forall x_2. \cdots \forall x_n.\, \varphi$ by $\forall X.\, \varphi$ and $\exists x_1.\, \exists x_2. \cdots \exists x_n.\, \varphi$ by $\exists X.\, \varphi$, accordingly, where $X = \{x_1, \ldots, x_n\}$.

Given a subset of variables $X \subseteq \mathscr{X}$, an *assignment* of X is a function $\alpha : X \to \mathbb{B}$ that maps each variable $x \in X$ to either true (1) or false (0). When the domain of α is not clear from context, we write α_X. A partial assignment $\beta : X \to \mathbb{B} \cup \{\bot\}$ may additionally set variables $x \in X$ to an undefined value $\bot$. We say that β is *compatible* with α, written $\beta \sqsubseteq \alpha$, if they have the same domains ($\mathrm{dom}(\alpha) = \mathrm{dom}(\beta)$) and $\alpha(x) = \beta(x)$ for all $x \in \mathrm{dom}(\alpha)$ where $\beta(x) \neq \bot$. For two assignments α and α' with domains $X = \mathrm{dom}(\alpha)$ and $X' = \mathrm{dom}(\alpha')$, we define the combination $\alpha \sqcup \alpha' : X \cup X' \to \mathbb{B}$ as $\alpha \sqcup \alpha'(x) = \alpha'(x)$ if

[1] In QBFEVAL'16, the best certifying QBF solver has solved less than half of the number of instances solved by the best non-certifying solver [31]. In the following two iterations of QBFEVAL up to this paper, certification has not been evaluated.

$x \in X'$ and $\alpha(x)$ otherwise. Note that α' overrides α for $x \in X \cap X'$. We define the *complement* $\overline{\alpha}$ to be $\overline{\alpha}(x) = \neg\alpha(x)$ for all $x \in \mathrm{dom}(\alpha)$. The complement of a partial assignment is defined analogously with $\neg\bot = \bot$. We denote by $\alpha \setminus X$ the assignment without the assignments for every $x \in X$, i.e., $\mathrm{dom}(\alpha \setminus X) = \mathrm{dom}(\alpha) \setminus X$. The *set of assignments* and *of partial assignments* of X is denoted by $\mathscr{A}(X)$ and $\mathscr{A}_\bot(X)$, respectively.

Example 1. Consider the assignments $\alpha = \{x \mapsto 0, y \mapsto 1\}$ and $\alpha' = \{x \mapsto 0, y \mapsto 0\}$ with $\mathrm{dom}(\alpha) = \mathrm{dom}(\alpha') = \{x, y\}$. For the partial assignment $\beta = \{x \mapsto \bot, y \mapsto 1\}$ it holds that $\beta \sqsubseteq \alpha$ and $\beta \not\sqsubseteq \alpha'$. Let $\alpha^* = \alpha \setminus \{y\} = \{x \mapsto 0\}$, then $\alpha \sqcup \overline{\alpha}^* = \{x \mapsto 1, y \mapsto 1\}$.

A quantifier $Qx.\,\varphi$ for $Q \in \{\exists, \forall\}$ *binds* the variable x in the *scope* φ. Variables that are not bound by a quantifier are called *free*. The set of free variables of formula φ is defined as $\mathit{free}(\varphi)$. The semantics of the satisfaction relation $\alpha \vDash \varphi$ is given as

$$
\begin{array}{ll}
\alpha \vDash x & \text{if } \alpha(x) = 1, \\
\alpha \vDash \neg\varphi & \text{if } \alpha \nvDash \varphi, \\
\alpha \vDash \varphi \vee \psi & \text{if } \alpha \vDash \varphi \text{ or } \alpha \vDash \psi, \\
\alpha \vDash \varphi \wedge \psi & \text{if } \alpha \vDash \varphi \text{ and } \alpha \vDash \psi, \\
\alpha \vDash \exists x.\,\varphi & \text{if some } \alpha' : \{x\} \to \mathbb{B} \text{ satisfies } \alpha \sqcup \alpha' \vDash \varphi, \text{ and} \\
\alpha \vDash \forall x.\,\varphi & \text{if all } \alpha' : \{x\} \to \mathbb{B} \text{ satisfy } \alpha \sqcup \alpha' \vDash \varphi.
\end{array}
$$

QBF satisfiability is the problem to determine, for a given QBF φ, the existence of an assignment α for the free variables of φ, such that the relation $\vDash$ holds.

An existentially quantified variable x *depends* on all universally quantified variables that are bound prior to x. A universally quantified variable x *depends* on all existentially quantified variables bound prior to x and additionally on the free variables. A free variable x *depends* on no variables. The set of dependencies of x is denoted by $\mathit{dep}(x)$. A Boolean function $f : \mathscr{A}(X) \to \mathbb{B}$ maps *assignments* of X to true or false. An assignment α over variables X can be identified by the conjunctive formula $\bigwedge_{x \in X \mid \alpha(x)=1} x \wedge \bigwedge_{x \in X \mid \alpha(x)=0} \neg x$. Similarly, Boolean functions can be represented by propositional formulas over the variables in their domain. Let $\varphi[f_{x_1}, \ldots, f_{x_n}]$ be the propositional formula where occurrences of x_i are replaced by the propositional representation of f_{x_i}. It is defined as

$$
\begin{aligned}
x[f_{x_1}, \ldots, f_{x_n}] &= \begin{cases} f_{x_i} & \text{if } x = x_i \text{ for some } i \\ x & \text{otherwise} \end{cases} \\
(\neg\varphi)[f_{x_1}, \ldots, f_{x_n}] &= \neg(\varphi[f_{x_1}, \ldots, f_{x_n}]) \\
(\varphi \vee \psi)[f_{x_1}, \ldots, f_{x_n}] &= (\varphi[f_{x_1}, \ldots, f_{x_n}]) \vee (\psi[f_{x_1}, \ldots, f_{x_n}]) \\
(\varphi \wedge \psi)[f_{x_1}, \ldots, f_{x_n}] &= (\varphi[f_{x_1}, \ldots, f_{x_n}]) \wedge (\psi[f_{x_1}, \ldots, f_{x_n}]) \\
(\exists x.\,\varphi)[f_{x_1}, \ldots, f_{x_n}] &= \varphi[f_{x_1}, \ldots, f_{x_n}] \\
(\forall x.\,\varphi)[f_{x_1}, \ldots, f_{x_n}] &= \varphi[f_{x_1}, \ldots, f_{x_n}]
\end{aligned}
$$

For example, let $\varphi = \forall x.\,\exists y.\,(x \vee \neg y) \wedge (\neg x \vee y)$ and let $f_y(x) = x$, then $\varphi[f_y] = (x \vee \neg x) \wedge (\neg x \vee x)$. A witness for a satisfiable QBF is a *Skolem function* $f_x : \mathscr{A}(\mathit{dep}(x)) \to \mathbb{B}$ for every variable x that is free or existentially quantified, such that $\neg\varphi[f_{x_1}, \ldots, f_{x_n}]$ is unsatisfiable. For unsatisfiable QBFs, the witnesses are defined dually and called *Herbrand functions*. We use the notation $\varphi[\alpha]$ to replace variables $x \in \mathrm{dom}(\alpha)$ by their assignments $\alpha(x)$.

A *closed* QBF is a formula without free variables. Closed QBFs are either true or false. A formula is in prenex form, if the formula consists of a quantifier prefix followed by a propositional formula. Every

QBF can be transformed into a closed QBF and into prenex form while maintaining satisfiability. A *literal* l is either a variable $x \in X$, or its negation $\neg x$. Given a set of literals $\{l_1, \ldots, l_n\}$, the disjunctive combination $(l_1 \vee \ldots \vee l_n)$ is called a *clause* and the conjunctive combination $(l_1 \wedge \ldots \wedge l_n)$ is called a *cube*. We denote by $var(l)$ the operation that returns the variable corresponding to l. A QBF is in negation normal form (NNF) if negation is only applied to variables. Every QBF can be transformed into NNF by at most doubling the size of the formula and without introducing new variables. For formulas in NNF, we treat literals as atoms.

3 Abstraction-based Algorithm

For QBFs given in CNF, there are recursive refinement algorithms where the refinement is based on clauses [23,32,35]. The underlying insight is that multiple variable assignments may lead to the satisfaction of the same clauses, hence, instead of communicating assignments, the information whether a clause is satisfied or not is communicated between quantifier blocks. Instead of excluding assignments one at a time, those algorithms may exclude multiple assignments with a single refinement step. In the following, we propose a generalization to formulas in negation normal form, i.e., we base the communication on the satisfaction of individual subformulas. For this section, we assume an arbitrary (closed, prenex) QBF $\Phi = QX_1 \ldots QX_n.\, \varphi$ with quantifier prefix $QX_1 \ldots QX_n$ and propositional body φ in NNF.

SAT solver. We use a generic solving function $\text{SAT}(\theta, \alpha)$ for propositional formula θ and assignment α, that returns whether $\theta \wedge \alpha$ is satisfiable. In the positive case, written $\text{SAT}(\theta, \alpha) \Rightarrow \text{SAT}(\alpha')$, it returns a satisfying assignment α'. We write $\text{SAT}(\theta, \alpha) \Rightarrow \text{SAT}(\alpha_V)$ if we are only interested in a subset V of the variables in θ. In the negative case, written $\text{SAT}(\theta, \alpha) \Rightarrow \text{UNSAT}(\beta)$, it returns a partial assignment $\beta \sqsubseteq \alpha$ such that $\theta \wedge \beta$ is unsatisfiable.

Example 2. We show a few examples of the usage of the SAT function using $\theta = (x \vee (\overline{x} \wedge y))$ and $\overline{\theta} = (\overline{x} \wedge (x \vee \overline{y}))$.

$$\begin{aligned}
&\text{SAT}(\overline{\theta}, \{\}) \Rightarrow \text{SAT}(\alpha_{\{x\}}) \qquad \text{where } \alpha_{\{x\}} = \{x \mapsto 0\} \\
&\text{SAT}(\theta, \alpha_{\{x\}}) \Rightarrow \text{SAT}(\alpha_{\{y\}}) \quad \text{where } \alpha_{\{y\}} = \{y \mapsto 1\} \\
&\text{SAT}(\overline{\theta}, \alpha_{\{x\}} \sqcup \alpha_{\{y\}}) \Rightarrow \text{UNSAT}(\{x \mapsto 0, y \mapsto 1\})
\end{aligned}$$

Notation. To facilitate working with arbitrary Boolean formulas, we start with introducing additional notation. Let $\mathscr{B}$ be the set of Boolean formulas and let $sf(\psi) \subset \mathscr{B}$ ($dsf(\psi) \subset \mathscr{B}$) be the set of (direct) subformulas of ψ (note that $\psi \in sf(\psi)$ but $\psi \notin dsf(\psi)$). For a propositional formula ψ, $type(\psi) \in \{lit, \vee, \wedge\}$ returns the Boolean connector if ψ is not a literal. For example, given $\psi = (x_1 \vee (\overline{x_1} \wedge x_2))$, $sf(\psi) = \{(x_1 \vee (\overline{x_1} \wedge x_2)), x_1, (\overline{x_1} \wedge x_2), \overline{x_1}, x_2\}$, $dsf(\psi) = \{x_1, (\overline{x_1} \wedge x_2)\}$, and $type(\psi) = \vee$.

Interface Variables. To communicate the value of subformulas, we introduce two special types of variables which we call *interface variables*. The value of those variables represents whether the value of a subformula is determined and in the positive case, the value itself. We only consider existential quantifiers as the definition for the universal quantifiers is dual with respect to negation. For a quantifier $\exists X$, we say that a subformula ψ is *positive* if ψ is conjunctive ($type(\psi) = \wedge$) and not falsified or ψ is disjunctive ($type(\psi) = \vee$) and satisfied. A subformula is *negative* if it is not positive. At quantifier $\exists X$, variable assignments determine whether a subformula ψ is positive or negative. The interface variables t_ψ and b_ψ represent whether ψ is positive, with the difference that t_ψ combines assignments from variables bound by outer quantifiers whereas b_ψ combines assignments from variables bound by outer quantifiers

and from variables X. We denote the set of variables t_ψ by T and call them T variables and analogously the set of variables b_ψ by B and call them B variables. For solving QX in the abstraction algorithm, we replace the communication of variable assignments by communicating assignments to T and B variables.

Abstractions. An *abstraction* for quantifier QX is a propositional formula θ_X over variables X, T, and B. The sets T_X and B_X contain those T and B variables that are used for communication at this quantifier level. Unlike previous approaches that utilize SAT solvers [22, 23, 32, 36], we keep for every quantifier level a *dual abstraction* $\overline{\theta}_X$ that is used for optimization of abstraction entries and translating interface variables. When translating a B variable to a T variable, we use the same index, e.g., in line 19 of Alg. 1, the B variables b_ψ are translated to T variables t_ψ of the inner quantifier. Before going into algorithmic details, we describe an execution of Algorithm 1 on Φ_{ex}.

Example 3. Consider the example $\Phi_{ex} = \forall x.\, \exists y.\, \overbrace{(x \vee \underbrace{(\overline{x} \wedge y)}_{\psi_2})}^{\psi_1}$ and its negation $\neg\Phi_{ex} = \exists x.\, \forall y.\, \overbrace{(\overline{x} \wedge \underbrace{(x \vee \overline{y})}_{\neg\psi_2})}^{\neg\psi_1}$. Assume that $\theta_x = b_1 \wedge (b_1 \rightarrow \overline{x}) \wedge (b_2 \rightarrow x)$ is the abstraction for quantifier $\forall x$ and that $\theta_y = (t_1 \vee b_2) \wedge (b_2 \rightarrow t_2) \wedge (b_2 \rightarrow y)$ is the abstraction for quantifier $\exists y$ (the definition of abstraction is given later).

The algorithm starts with the top level quantifier $\forall x$. The universal player has to set x to false to satisfy θ_x, leading to the unique assignment $\alpha_B = \{b_1 \mapsto 1, b_2 \mapsto 0\}$ of B variables. This means that $\neg\psi_1$ is positive and $\neg\psi_2$ is negative in $\neg\Phi_{ex}$. Due to duality, ψ_1 is negative and ψ_2 is positive in Φ_{ex}. Thus, the assignment α_B is translated into assignment α_T of T variables in θ_y by negation ($\alpha_T = \{t_1 \mapsto 0, t_2 \mapsto 1\}$, line 19). At the existential quantifier $\exists y$, the abstraction θ_y is solved under the assumption α_T, resulting in a satisfiable query with variable assignment $\{y \mapsto 1\}$. We then use the dual abstraction $\overline{\theta}_y = b_1 \wedge (b_1 \rightarrow t_1) \wedge (b_1 \rightarrow b_2) \wedge (b_2 \rightarrow t_2 \vee \overline{y})$ to translate α_T to a partial assignment β_T (line 4). The partial assignment $\{t_2 \mapsto 0, y \mapsto 1\}$ is enough to falsify the dual abstraction $\overline{\theta}_y$, thus, the assumption $\alpha_T(t_2) = 1$ is needed to satisfy θ_y and the partial assignment β_T with $\beta(t_2) = 1$ is returned (line 4). The following refinement forces that b_2 must be set to true in the next iteration, i.e., $\theta_x = \theta_x \wedge b_2$. This depletes all possible assignments of the universal quantifier, thus, proving that the instance is true.

Algorithm. The main procedure of Algorithm 1 is ABSTRACTION-QBF-REC, that recurses on the quantifier prefix. In line 2, a candidate solution is generated (represented by an assignment α_{B_X}) with respect to an assignment α_{T_X} given by the outer quantifier. In the following, the candidate solution is verified recursively (line 6) and in the negative case the abstraction θ_X is refined by a blocking clause (line 10) that eliminates (at least) this candidate.

To verify a candidate α_{B_X} recursively, it is *translated* into an assignment α_{T_Y} of the inner quantifier $\overline{Q}Y$ (line 19). This is done by negation since a subformula ψ is positive for QX iff it is negative for $\overline{Q}Y$. The *refinement* operation generates, given a counterexample represented as a partial assignment β_{T_Y}, a clause consisting of B variables that excludes this counterexample: For every T variable t_ψ that is contained in the counterexample and is positive for the inner quantifier, the refinement adds a B variable b_ψ meaning that one of those subformulas must be positive for QX in the next iteration.

The dual abstraction $\overline{\theta}_X$ is defined as the abstraction for $\overline{Q}X$ and is used in two ways. First, it optimizes that candidate α_{B_X} in the propositional case (line 4), i.e., it generates potentially smaller witnesses. Second, it translates an assignment of T_Y variables β_{T_Y} to an assignment of T_X variables β_{T_X} that is returned to the outer quantifier (line 9).

We now focus on the abstraction θ_X. In Example 3, we have already seen an instance of the abstraction that we formally introduce in the following. The abstraction is a modification of the Plaisted-Greenbaum encoding [30]: for subformula ψ, the b-literal b_ψ corresponds to the defining literal of the Plaisted-Greenbaum encoding (see definition of *enc* in Fig. 1). $enc_\psi(\psi')$ is responsible for abstracting

Algorithm 1 Abstraction Based Algorithm

1: **procedure** ABSTRACTION-QBF-REC($QX.\,\varphi, \alpha_{T_X}$)
2: **while** SAT(θ_X, α_{T_X}) $\Rightarrow$ SAT($\alpha_X \sqcup \alpha_{B_X}$) **do** ▷ generate candidate α_{B_X}
3: **if** φ is propositional **then**
4: **return** $\langle \text{SAT}_Q, \text{DUAL-OPT}(\alpha_X, \alpha_{T_X})\rangle$
5: $\alpha_{T_Y} \leftarrow$ TRANSLATE(X, φ, α_{B_X})
6: $\langle result, \beta_{T_Y}\rangle \leftarrow$ ABSTRACTION-QBF-REC(φ, α_{T_Y}) ▷ verify α_{T_Y} recursively
7: **if** $result = \text{SAT}_Q$ **then**
8: $\overline{\theta}_X \leftarrow \overline{\theta}_X \wedge \text{REFINE}_X(\beta_{T_Y})$ ▷ refine dual abstraction
9: **return** $\langle \text{SAT}_Q, \text{DUAL-OPT}(\alpha_X, \alpha_{T_X})\rangle$
10: $\theta_X \leftarrow \theta_X \wedge \text{REFINE}_X(\beta_{T_Y})$ ▷ refine abstraction
11: let β_{T_X} be the failed assumptions (SAT(θ_X, α_{T_X}) $\Rightarrow$ UNSAT(β_{T_X}))
12: **return** $\langle \text{UNSAT}_Q, \beta_{T_X}\rangle$ ▷ $\beta_{T_X} \sqsubseteq \alpha_{T_X}$
13: **procedure** REFINE$_X$(β_{T_Y})
14: **return** $\bigvee_{b\in B} b$ **where** $B = \{b_\psi \in B_X \mid \beta_{T_Y}(t_\psi) = 1\}$
15: **procedure** DUAL-OPT(α_X, α_{T_X})
16: SAT($\overline{\theta}_X, \alpha_X \sqcup \overline{\alpha}_{T_X}$) $\Rightarrow$ UNSAT(β_{T_X})
17: **return** $\overline{\beta}_{T_X}$ ▷ $\overline{\beta}_{T_X} \sqsubseteq \alpha_{T_X}$
18: **procedure** TRANSLATE(X, $Q\,Y.\,\varphi$, α_{B_X})
19: **return** α_{T_Y} s.t. $\alpha_{T_Y}(t_\psi) = \overline{\alpha}_{B_X}(b_\psi)$ for all $t_\psi \in T_Y$ ▷ translate $\alpha_{B_X} \to \alpha_{T_Y}$
20: **procedure** ABSTRACTION-QBF($QX_1 \ldots QX_n.\,\varphi$)
21: **for all** QX_i, initialize θ_{X_i} and $\overline{\theta}_{X_i}$
22: **return** ABSTRACTION-QBF-REC($QX_1 \ldots QX_n.\,\varphi, \{\}$)

from actual assignments: literals bound at the quantifier are returned unchanged, literals bound at an outer quantifier are abstracted as a T variable, and we use the defining literal $b_{\psi'}$ of a subformula ψ' if the valuation of the subformula is guaranteed to be fixed, i.e., there is no inner influence.

Given a propositional formula φ in NNF and a quantifier $\exists X$, we build the following propositional formula in CNF representing the abstraction $\theta_X = out_\varphi(\varphi) \wedge \bigwedge_{\psi\in sf(\varphi)\wedge type(\psi)\neq lit} enc(\psi)$ for this quantifier, where *out* encodes that φ must hold and *enc* defines a CNF formula that encodes the truth of subformula ψ with respect to the valuations of the current, inner, and outer quantifier represented by B and T variables, respectively. The definitions are given in Fig. 1. The abstraction of a quantifier $\forall X$ is defined as the existential abstraction for $\neg\varphi$. Note that not every B literal that is used in the abstraction may be exposed as an interface literal. For the given abstraction, we define the set of interface variables for quantifier QX_i as

$$B_{X_i} = \{b_\psi \mid \psi \in sf(\varphi) \wedge \psi \text{ contains a variable bound} \leq i\}, \text{ and}$$
$$T_{X_i} = \{t_\psi \mid \psi \in sf(\varphi) \wedge \psi \text{ contains a variable bound} < i\}.$$

Theorem 1. ABSTRACTION-QBF *is sound and complete.*

The proof is given in Section 5 and relies on techniques developed for certification in the next section.

$$enc(\psi) = \begin{cases} \bigwedge_{\substack{\psi' \in dsf(\psi) \\ enc_\psi(\psi') \neq \bot}} \left(b_\psi \rightarrow enc_\psi(\psi')\right) & \text{if } type(\psi) = \wedge \\ b_\psi \rightarrow \bigvee_{\substack{\psi' \in dsf(\psi) \\ enc_\psi(\psi') \neq \bot}} enc_\psi(\psi') & \text{if } type(\psi) = \vee \end{cases}$$

$$enc_\psi(\psi') = \begin{cases} \psi' & \text{if } type(\psi') = lit \wedge var(\psi') \in X \\ t_\psi & \text{if } type(\psi') = lit \wedge \text{literal is bound by outer quantifier} \\ b_{\psi'} & \text{if } type(\psi') \neq lit \wedge \psi' \text{ has no inner influence} \\ \bot & \text{otherwise} \end{cases}$$

$$out_\psi(\psi') = \begin{cases} b_{\psi'} & \text{if } type(\psi') = \wedge \\ \bigvee_{\psi^* \in dsf(\psi')} out_{\psi'}(\psi^*) & \text{if } type(\psi') = \vee \\ \psi' & \text{if } type(\psi') = lit \wedge var(\psi') \in X \\ t_\psi & \text{if } type(\psi') = lit \wedge \text{literal is bound by outer quantifier} \\ \neg b_\psi & \text{if } type(\psi') = lit \wedge \text{literal is bound by inner quantifier} \end{cases}$$

Figure 1: Definition of the abstraction for quantifier block $\exists X$.

4 Certification

Certification is an essential component of QBF solving. Certification amounts to extracting witnessing functions from a QBF, either *Skolem* functions for true QBFs or *Herbrand* functions for false QBFs. Not only does it allow to verify the solver result, but the resulting functions can also be used in the context of the application. The main result of this section is a proof format for our abstraction algorithm and an efficient algorithm to transform proof traces into Boolean functions.

Proof Format. To extract a witness from a run of ABSTRACTION-QBF, we need to remember *situations* and *reactions*, represented by assignments to T and X, that were satisfiable for the respective quantifier. Hence, the proof $\mathscr{P}$ consists of a sequence of pairs $\langle \beta_T, \alpha_X \rangle \in (\mathscr{A}_\bot(T) \times \mathscr{A}(X))$ and these pairs can be obtained from the algorithm by the result β_{T_X} of the query to the dual abstraction $\overline{\theta}_X$ in line 16. As an immediate consequence, the number of pairs in the proof trace is linear in the number of iterations of the algorithm. We define a function $\mathscr{C}_X : \mathscr{A}_\bot(T) \rightarrow \mathscr{B}(V)$ which, for a given quantifier QX, maps an assignment β_T to a Boolean formula over variables V bound by outer quantifiers (with respect to X). Intuitively, $\mathscr{C}_X(\beta_T)$ describes those assignments that lead to β_T in the abstraction of quantifier QX.

Function Extraction. Prior to the function extraction, we filter out those pairs from the proof $\mathscr{P}$ that correspond to the variables that are dependencies and do not describe a function, i.e., universal variables for true QBFs and existential variables for false QBFs. The remaining proof consists of pairs $\langle \beta_T, \alpha_X \rangle$ where $\mathscr{C}_X(\beta_T)$ is a formula that represents the situation where the response α_X is correct. Let $\langle \beta_T^1, \alpha_X^1 \rangle \dots \langle \beta_T^n, \alpha_X^n \rangle$ be the pairs corresponding to quantifier QX and let $x \in X$ be some variable, the

function $f_x : \mathscr{A}(dep(x)) \to \mathbb{B}$ is defined as

$$f_x \equiv \bigvee_{i=1}^{n} \left((\alpha_X^i(x) = 1) \wedge \mathscr{C}_X(\beta_T^i) \wedge \bigwedge_{j<i} \neg \mathscr{C}_X(\beta_T^j) \right) \tag{1}$$

This construction is similar to previous extraction algorithms, including [2, 32]. The definition of $\mathscr{C}_X$ allows that f_x may depend on variables in outer quantifiers corresponding to functions instead of dependencies. By replacing those variables with their extracted functions, one can make sure that f_x depends only on $dep(x)$. The size of f_x, measured in terms of distinct subformulas, is linear in the number of pairs and, hence, linear in the size of the proof.

Theorem 2. *Given a QBF Φ and proof trace $\mathscr{P}$, the runtime of the function extraction algorithm is in $\mathscr{O}(|\mathscr{P}|)$. The size of the resulting functions is linear in the size of $\mathscr{P}$.*

Certification. A *certificate* is a representation of all functions that, combined, witness the result of the QBF. A certificate is correct, if two conditions are satisfied: the certificate is (1) functionally correct and (2) well-formed. Functional correctness can be checked by a propositional SAT query to $\neg\varphi$ (respectively φ for false QBFs) where every occurrence of a function variable y is replaced by the function f_y. The unsatisfiability of this query witnesses functional correctness. The well-formedness criterion concerns the representation of the certificate, usually as a circuit, and requires that the representation of a function depends only on its dependencies. One can further differentiate *syntactical* and *semantical* well-formedness. A certificate is syntactically ill-formed if a non-dependency is reachable from the output of a function. A certificate is semantically ill-formed if a valuation change of a set of non-dependencies changes the valuation of a function. Our function extraction guarantees syntactical well-formedness and therefore any further circuit simplification guarantees at least semantical well-formedness.

Example 4. Consider again our example $\Phi_{ex} = \forall x.\, \exists y.\, \overbrace{(x \vee \underbrace{(\bar{x} \wedge y)}_{\psi_2})}^{\psi_1}$. It holds that $\mathscr{C}_{\{y\}}(\{t_1 \mapsto 1\}) = x$ and $\mathscr{C}_{\{y\}}(\{t_2 \mapsto 1\}) = \bar{x}$, because setting x to true satisfies ψ_1 and setting it to false does not falsify ψ_2. The proof trace for Φ_{ex} is $\langle \{t_2 \mapsto 1\}, \{y \mapsto 1\} \rangle$ (see Example 3) and the resulting Skolem function is $f_y(x) \equiv \mathscr{C}_{\{y\}}(\{t_2 \mapsto 1\}) \equiv \bar{x}$. To verify f_y, we check $\neg\Phi_{ex}[f_y] = \exists x.\, (\bar{x} \wedge (x \vee \bar{\bar{x}}))$ for unsatisfiability.

5 Correctness

In the following, we formalize properties of the abstraction and prove the algorithm correct. To relate variable assignments and assignments of T variables, we use the function $\mathscr{C}_X$ which is defined in the previous section. $\mathscr{C}_X(\beta_T)$ is a propositional formula over the outer variables V (with respect to X) that describes the assignments leading to β_T in the abstraction of quantifier QX. An assignment α_V is *compatible* with a partial assignment β_{T_X}, if it satisfies $\mathscr{C}_X(\beta_{T_X})$. Then, we write $\alpha_V \prec \beta_{T_X}$ for short.

The proof of Theorem 1 is done by induction on the structure of the quantifier prefix. To prove the base case of the induction, Lemma 1.2 states that a satisfiable result in the innermost quantifier corresponds to satisfaction and falsification of the propositional formula for the existential and universal player, respectively (see line 4 of Algorithm 1). Further, Lemma 1.3 states the correctness of early termination, i.e., if the initial abstraction returns unsatisfiable, the propositional formula is unsatisfiable under the current assignment (dual for universal player).

Lemma 1. *The abstraction has the following properties:*

1. *For a quantifier alternation $QX.\overline{Q}Y$, the set of outer B literals B_X matches the set of inner T literals T_Y, i.e., $\{\psi \mid b_\psi \in B_X\} = \{\psi \mid t_\psi \in T_Y\}$.*
2. *If θ_X is satisfiable under assumptions α_{T_X} where X is the innermost quantifier, then for all assignments α with $\alpha \prec \alpha_{T_X}$, there is an assignment α^* with $\alpha \sqsubseteq \alpha^*$ such that $\Phi[\alpha^*]$ is true ($Q = \exists$), respectively false ($Q = \forall$).*
3. *If θ_X is unsatisfiable under assumptions β_{T_X}, then for all assignments α with $\alpha \prec \beta_{T_X}$ it holds that $\Phi[\alpha]$ is false if $Q = \exists$, respectively true if $Q = \forall$ (dual for $\overline{\theta}_X$).*

Proof. 1. Holds by definition of T_X and B_X (Section 3).

2. The B variables at the innermost level correspond to the auxiliary variables in the encoding due to Plaisted and Greenbaum [30]. Further, all outer quantified variables are replaced by T variables. Both properties together show that the claim holds.
3. For the innermost abstraction, this claim holds by the same argument as in (2). For the other abstractions, note that the formula θ_X is weaker than the Plaisted-Greenbaum encoding: the encoding $enc_\psi(\psi')$ of a subformula ψ only takes other subformulas ($type(\psi') \neq lit$) into account if ψ' is not influenced by a variable bound by an inner quantifier. □

We now have all tools available to prove Theorem 1. The first two invariants state that Lemma 1.3 holds during the execution of the algorithm, that is, also after the refinement steps. The last two invariants connect variable assignments to the result of the recursive call of ABSTRACTION-QBF-REC.

Proof of Theorem 1. ABSTRACTION-QBF-REC maintains the following invariants that witness the correctness of ABSTRACTION-QBF.

1. If θ_X is unsatisfiable under assumptions β_{T_X}, then for all assignments α with $\alpha \prec \beta_{T_X}$ it holds that $\Phi[\alpha]$ is false if $Q = \exists$, respectively true if $Q = \forall$.
2. If $\overline{\theta}_X$ is unsatisfiable under assumptions β_{T_X}, then for all assignments α with $\alpha \prec \beta_{T_X}$ it holds that $\Phi[\alpha]$ is true if $Q = \exists$, respectively false if $Q = \forall$.
3. If ABSTRACTION-QBF-REC returns $\langle \text{SAT}, \beta_{T_X} \rangle$, then for all α with $\alpha \prec \beta_{T_X}$ it holds that $\Phi[\alpha]$ is true.
4. If ABSTRACTION-QBF-REC returns $\langle \text{UNSAT}, \beta_{T_X} \rangle$, then for all α with $\alpha \prec \beta_{T_X}$ it holds that $\Phi[\alpha]$ is false.

Claim (1) and (2) hold initially by Lemma 1.3.

Base case. ABSTRACTION-QBF-REC$(\exists X.\, \varphi, \alpha_{T_X})$ where φ is propositional (case $\forall$ is dual). Assume $\text{SAT}(\theta_X, \alpha_{T_X})$ is unsatisfiable (line 2) with failed assumptions β_{T_X}. Then by (1) for all assignments α with $\alpha \prec \beta_{T_X}$, $\Phi[\alpha]$ is false, proving (4). Assume $\text{SAT}(\theta_X, \alpha_{T_X})$ is satisfiable (line 2), then by Lemma 1.2 for all $\alpha^* \prec \alpha_{T_X}$, there is an assignment α with $\alpha \sqsubseteq \alpha^*$ such that $\Phi[\alpha]$ is true. Together with Lemma 2 this proves (3).

Induction step. ABSTRACTION-QBF-REC$(\exists X. \forall Y \ldots Q X_n.\, \varphi, \alpha_{T_X})$ (case $\forall$ is dual). Assume that $\text{SAT}(\theta_X, \alpha_{T_X})$ is unsatisfiable (line 2) with failed assumptions β_{T_X}. Then by (1) for all assignments α^* with $\alpha^* \prec \beta_{T_X}$, $\Phi[\alpha^*]$ is false, proving (4). Assume $\text{SAT}(\theta_X, \alpha_{T_X})$ is satisfiable (line 2). The candidate α_{B_X} is translated into an assignment α_{T_Y} (Lemma 1.1). The following recursive call (line 6) returns either SAT or UNSAT. If the result is $\langle \text{SAT}, \beta_{T_Y} \rangle$, then by IH and claim (3), for all α^* with $\alpha^* \prec \beta_{T_Y}$ it holds that $\Phi[\alpha^*]$ is true. Excluding β_{T_Y} from $\overline{\theta}_X$ (line 8) thus preserves invariant (2). Using invariant (2) and Lemma 2, this proves (3) when returning from line 9. If the result is $\langle \text{UNSAT}, \beta_{T_Y} \rangle$, then by IH and

Table 1: This table shows the number of solved instances within 10 minutes.

(a) QBFEVAL'18

Solver	Total	Sat	Unsat	Unique
QuAbS	181	82	99	1
cQESTO	160	75	85	1
GhostQ	157	69	88	0
QFUN	139	74	65	5
Qute	116	42	74	0

(b) Petri Game Benchmarks

Solver	Total	Sat	Unsat	Unique
QuAbS	195	123	72	14
cQESTO	189	127	62	11
QFUN	141	90	51	0
GhostQ	139	85	54	0
Qute	100	64	36	0

claim (4), for all α^* with $\alpha^* \prec \beta_{T_Y}$ it holds that $\Phi[\alpha]$ is false. Excluding β_{T_Y} from θ_X (line 10) preserves invariant (1). Completeness (the while loop cannot execute infinitely often) follows from the fact that there are only finitely many different blocking clauses. □

Lemma 2. *Let θ_X and let α_{T_X} be given. If α_X is a satisfying assignment of $\theta_X[\alpha_{T_X}]$, then* $\text{SAT}(\overline{\theta}_X, \alpha_X \sqcup \overline{\alpha}_{T_X})$ *returns* $\text{UNSAT}(\beta_{T_X})$ *and for all α^* with $\overline{\beta}_{T_X} \sqsubseteq \alpha^*$, $\theta_X[\alpha^* \sqcup \alpha_X]$ is true.*

Proof. Note that by definition of θ_X and $\overline{\theta}_X$, an assignment α_{T_X} in θ_X corresponds to an assignment $\overline{\alpha}_{T_X}$ in the dual abstraction $\overline{\theta}_X$, i.e, α_{T_X} and $\overline{\alpha}_{T_X}$ represent the same variable assignments (outer variables w.r.t. QX) in θ_X and $\overline{\theta}_X$, respectively. As α_X is a satisfying assignment for $\theta_X[\alpha_{T_X}]$, $\overline{\theta}_X[\overline{\alpha}_{T_X} \sqcup \alpha_X]$ is unsatisfiable. By definition of failed assumptions, $\beta_{T_X} \sqsubseteq \overline{\alpha}_{T_X}$ and $\text{SAT}(\overline{\theta}_X, \beta_{T_X})$ returns UNSAT, i.e., there is no α with $\beta_{T_X} \sqsubseteq \alpha$ that satisfies $\overline{\theta}_X$, hence, all α^* with $\overline{\beta}_{T_X} \sqsubseteq \alpha^*$ satisfy $\theta_X[\alpha_X]$. □

6 Evaluation

We implemented Algorithm 1 and its optimizations in a solver called QuAbS[2] (Quantified Abstraction Solver) that takes QBFs in the standard format QCIR. As the underlying SAT solver, we use CryptoMiniSat [33]. We compare QuAbS against the publicly available QBF solvers that support the QCIR format, namely GhostQ [25], QFUN [21], cQESTO [20], and Qute [28]. For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon processor and 32 GB of memory. The timeout and memout were set to 10 minutes and 8 GB, respectively.

QuAbS has been independently evaluated in the annual QBF competition, called *QBFEVAL* and the results of the latest evaluation are given in Table 1(a). Notably, QuAbS solved most instances, 21 more than the second best solver. The certification capabilities of QuAbS are used in the reactive synthesis tool BoSy [6], which won the synthesis track in the reactive synthesis competition (SYNTCOMP) 2016 and 2017 [17, 19].

Certification. We implemented the certification approach described in Section 4, but instead of generating proof traces, we build the certificates (represented by And-Inverter Graphs) within the solving loop. This enables building Skolem and Herbrand functions in parallel during solving and minimizes the certification overhead. In the verification step, we use CryptoMiniSat to solve the functional correctness query. The size of a certificate is measured as the number of AND gates.

We evaluate the certification capabilities of QuAbS on synthesis benchmark sets that are designed to take advantage of the structural problem definition. The *petri-games* benchmark set uses the bounded

[2]Source code available at `https://github.com/ltentrup/quabs`

Table 2: Result of the certification run with timeout of 10 minutes for solving and verification, respectively. The average size of the certificate and the accumulated time spend on solving and verification are restricted to verified instances.

Benchmark set	#solved	#verified	avg. size	solving [sec.]	verification [sec.]
petri-games	136	120	90,699	5389	3508
safety-synt	160	144	41,125	153	2569
bounded-synthesis	339	339	11,390	4552	1457
tree-models	186	179	49,456	4032	9528

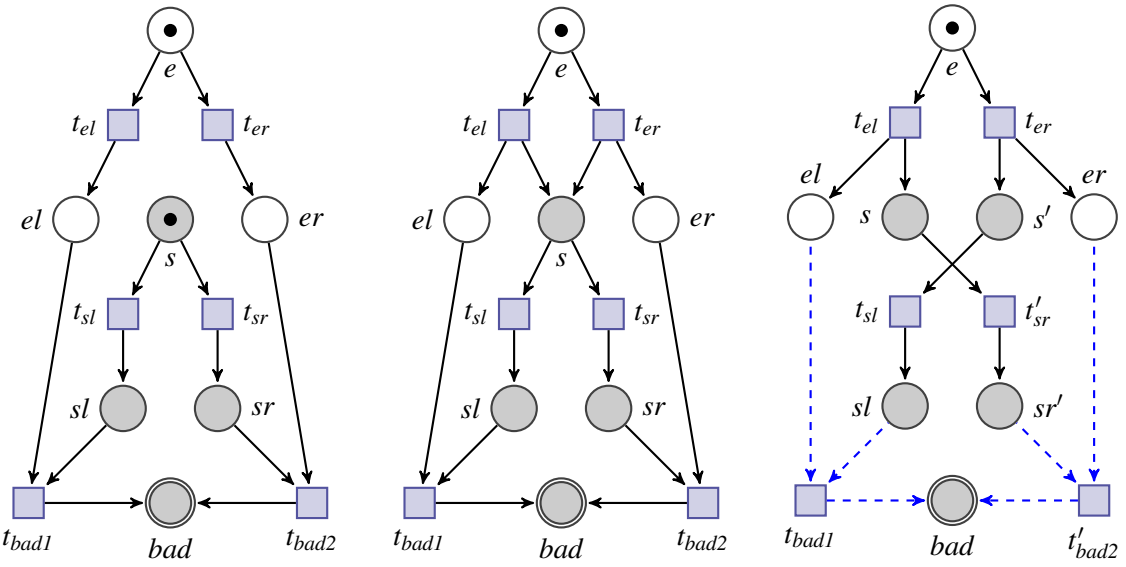
synthesis approach for Petri games [8, 10]. The *safety-synt* benchmark set was created from the safety benchmarks of SYNTCOMP 2014 [18]. The *bounded-synthesis* benchmark set was created from the tool BoSy [6] using the QBF encoding of the reactive synthesis problem using LTL specifications [5]. The *tree-models* benchmark set was created from LTL benchmarks of SYNTCOMP 2016 [19]. All those benchmarks have in common that it is possible to directly build implementations from satisfiable queries.

The overall effect of the certification approach on the runtimes is negligible (less than 1% increase) which we consider as achieving our goal that the combination of solving and certification can be implemented efficiently. Table 2 shows the results of the certification run. The number of verified instances is lower than the number of solved ones because the verifier exceeded the time- and memory-limit on some instances that could be solved within the limits. To further reduce the size of certificates, one can employ circuit minimization techniques. Especially compared to CNF certification, these results are very promising and could boost the use of QBF in synthesis applications.

6.1 Case Study: Petri Games

In this case study, we outline how the certification capabilities of QuAbS can be used for the analysis of unrealizable Petri games and for the construction of implementations from winning strategies. Petri games [9,10] represent the synthesis problem for distributed, asynchronous systems with causal memory. The QBF encoding [7,8] of those games is particularly challenging for CNF solvers: hardly any instance can be solved, even with enabled preprocessing and independent of the used solver, ruling out existing CNF certification approaches. In contrast, non-CNF solvers scale much better as shown in Table 1(b), with QuAbS performing best overall.

Distributed Synthesis of Asynchronous Systems. The manual implementation of programs is a tedious and error-prone task. The automatic synthesis of a correct implementation for a given specification can help the developer to focus on what requirements to fulfill instead of how to fulfill them. The intricate communication of asynchronous processes in distributed systems would greatly benefit from the automatic synthesis of correct implementations for each process. *Petri games* define the synthesis problem of asynchronous, distributed systems with causal memory. The system is *distributed* in the sense that its consists of local processes with individual strategies without global controller. The system is *asynchronous* in the sense that local processes advance at individual pace and no global clock exists at which processes produce outputs. Local strategies at a process can utilize *causal memory* which only allows processes to exchange information upon synchronization. Petri games are based on an underlying Petri net which makes it possible to utilize the unfolding as representation of causal memory. The simplest winning condition for Petri games are bad places which the system has to avoid while the environment tries to reach such places.



(a) A Petri game where the system should not mimic the environment's behavior but no communication takes place prior to the system's decisions.

(b) The environment forwards its decision to the system and afterwards the system should not mimic this decision.

(c) A winning strategy where the system does not mimic the environment such that transitions to the bad place (dashed in blue) become unreachable.

Figure 2: An example workflow of designing a Petri game is outlined. QUABS produces counterexamples to any strategy in the left Petri game. From there, it becomes clear that there is no information exchange between the system and the environment. Therefore, the design of the Petri game is changed to the one in the middle where the environment leaks its decision to the system. For this game, we can extract the winning strategy on the right using QUABS which avoids the bad place as the system answers with opposite decisions to the decisions of the environment.

Consider the example Petri game from Fig. 2(a) where the system and the environment can both decide between left and right transitions and the bad place can only be avoided by opposite decisions. Petri games are an extension of Petri nets where the places are distributed to either belong to the system (gray places) or to the environment (white places). The tokens flowing through the underlying net now represent players depending on the type of place they are residing in: strategies of system players can restrict which outgoing transitions are allowed to fire whereas environment players decide the flow of tokens in the net. In the game of Fig. 2(a), the choice of system and environment are *independent*, i.e., the system player has no strategy to avoid the bad place: choosing either the left (t_{sl}) or right (t_{sr}) transition, the environment will do the same, leading the game to the bad place.

Strategy Construction and Strategy Refutation. As the Petri game in Fig. 2(a) has no winning strategy, the QBF encoding [7] is unsatisfiable and QUABS returns a certificate for the universal player. This certificate represents a flow of tokens leading to the bad state for *every* system strategy. When the system only decides to enable t_{sl} and to not enable t_{sr} then one counterexample moves the environment token from e to el, the system token from s to sl, and afterwards fires the transition to reach the bad place. An analog counterexample is returned when the system enables t_{sr} and does not enable t_{sl}. When the system enables neither transition then the counterexample moves the environment token from e to el and then reaches a deadlock without termination. This situation is forbidden for strategies as otherwise the win-

ning condition of avoiding bad places would be a trivial. When the system activates both transitions then already the initial marking constitutes a counterexample as the system's decision is non-deterministic.

From these counterexamples, we can derive that we have to introduce communication between the system and the environment. The easiest way to do so is given in Fig. 2(b) where the system player is created with the decision of the environment and then can only afterwards react to it. The different causal memory of the system player in s depending on whether t_{el} or t_{er} was fired results in the unfolding of s (indicated by $'$), as depicted in Fig. 2(c). Then, a winning strategy exists where the system player makes a different decision to the previous environment decision. The satisfying assignment of QuAbS in the QBF encoding of this problem allows to directly remove not activated transitions (t_{sr} and t'_{sl}) and their resulting unreachable parts of the game, making all transitions to the bad place unreachable (indicated as dashed blue lines in Fig. 2(c)).

7 Related Work

Other QBF solving techniques that use structural information are conceptually very different, such as DPLL like [4, 14, 25, 28] and expansion [21, 22, 26, 29]. We extend work on QBF solving techniques that communicate the satisfaction of clauses through a recursive refinement algorithm [23, 32, 35] that were limited to conjunctive normal form. Further, the maintenance of a dual abstraction for optimization is new in this context and the certification approach is different and, as shown in the evaluation, much more efficient than the one presented for CAQE [32]. The structure of independent quantifiers in non-prenex formulas can be used for parallelization during solving for this kind of algorithms [34]. cQESTO [20] is a recently introduced circuit solver based on a similar algorithm as presented in this paper. The algorithm, however, differs in the way abstractions are built: we produce a "static" abstraction upfront and learn subformula valuations during solving, while cQESTO evaluates the circuit under the current variable assignments and re-encodes the resulting partial circuit using the Tseitin transformation in each refinement step. To our knowledge, cQESTO cannot produce certificates. Certification has been considered in the context of CNF solving techniques [1, 16, 27, 32] but we are not aware of another work considering certification in the more general setting. The duality of circuit based QBF solving has been used to enhance search based CNF solvers [13, 15] but this is different to our use of a dual abstraction during solving.

8 Conclusion

We presented a QBF solving algorithm that exploits the structure in the propositional formula. Further, we defined a certification format suitable for this algorithm and described an efficient algorithm to extract solution witnesses from true, respectively false, QBFs. We have implemented the solving and certification techniques in a tool called QuAbS which won the QBF competition QBFEVAL'18. We have achieved our goal of the certification approach having nearly no overhead over pure solving approaches. For the case study of Petri games, we outlined how the certification techniques of QuAbS allow the analysis of unrealizable Petri games and the construction of implementations for realizable Petri games.

Acknowledgments

We thank Mikolás Janota for reporting a problem with an earlier formulation of the abstraction and the anonymous reviewers for their helpful comments.

References

[1] Valeriy Balabanov & Jie-Hong R. Jiang (2012): *Unified QBF certification and its applications*. *Formal Methods in System Design* 41(1), pp. 45–65, doi:10.1007/s10703-012-0152-6.

[2] Olaf Beyersdorff, Ilario Bonacina & Leroy Chew (2016): *Lower Bounds: From Circuits to QBF Proof Systems*. In: *Proceedings of ITCS*, ACM, pp. 249–260, doi:10.1145/2840728.2840740.

[3] Roderick Bloem, Robert Könighofer & Martina Seidl (2014): *SAT-Based Synthesis Methods for Safety Specs*. In: *Proceedings of VMCAI*, *LNCS* 8318, Springer, pp. 1–20, doi:10.1007/978-3-642-54013-4_1.

[4] Uwe Egly, Martina Seidl & Stefan Woltran (2009): *A solver for QBFs in negation normal form*. *Constraints* 14(1), pp. 38–79, doi:10.1007/s10601-008-9055-y.

[5] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe & Leander Tentrup (2017): *Encodings of Bounded Synthesis*. In: *Proceedings of TACAS*, *LNCS* 10205, pp. 354–370, doi:10.1007/978-3-662-54577-5_20.

[6] Peter Faymonville, Bernd Finkbeiner & Leander Tentrup (2017): *BoSy: An Experimentation Framework for Bounded Synthesis*. In: *Proceedings of CAV*, *LNCS* 10427, Springer, pp. 325–332, doi:10.1007/978-3-319-63390-9_17.

[7] Bernd Finkbeiner (2015): *Bounded Synthesis for Petri Games*. In: *Proceedings of Correct System Design*, *LNCS* 9360, Springer, pp. 223–237, doi:10.1007/978-3-319-23506-6_15.

[8] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch & Ernst-Rüdiger Olderog (2017): *Symbolic vs. Bounded Synthesis for Petri Games*. In: *Proceedings of SYNT@CAV*, *EPTCS* 260, pp. 23–43, doi:10.4204/EPTCS.260.5.

[9] Bernd Finkbeiner, Manuel Gieseking & Ernst-Rüdiger Olderog (2015): *Adam: Causality-Based Synthesis of Distributed Systems*. In: *Proceedings of CAV*, *LNCS* 9206, Springer, pp. 433–439, doi:10.1007/978-3-319-21690-4_25.

[10] Bernd Finkbeiner & Ernst-Rüdiger Olderog (2017): *Petri games: Synthesis of distributed systems with causal memory*. *Inf. Comput.* 253, pp. 181–203, doi:10.1016/j.ic.2016.07.006.

[11] Bernd Finkbeiner & Leander Tentrup (2014): *Detecting Unrealizable Specifications of Distributed Systems*. In: *Proceedings of TACAS*, *LNCS* 8413, Springer, pp. 78–92, doi:10.1007/978-3-642-54862-8_6.

[12] Bernd Finkbeiner & Leander Tentrup (2015): *Detecting Unrealizability of Distributed Fault-tolerant Systems*. *Logical Methods in Computer Science* 11(3), doi:10.2168/LMCS-11(3:12)2015.

[13] Alexandra Goultiaeva & Fahiem Bacchus (2010): *Exploiting QBF Duality on a Circuit Representation*. In: *Proceedings of AAAI*, AAAI Press.

[14] Alexandra Goultiaeva, Vicki Iverson & Fahiem Bacchus (2009): *Beyond CNF: A Circuit-Based QBF Solver*. In: *Proceedings of SAT*, *LNCS* 5584, Springer, pp. 412–426, doi:10.1007/978-3-642-02777-2_38.

[15] Alexandra Goultiaeva, Martina Seidl & Armin Biere (2013): *Bridging the gap between dual propagation and CNF-based QBF solving*. In: *Proceedings of DATE*, IEEE, pp. 811–814, doi:10.7873/DATE.2013.172.

[16] Marijn Heule, Martina Seidl & Armin Biere (2014): *Efficient extraction of Skolem functions from QRAT proofs*. In: *Proceedings of FMCAD*, IEEE, pp. 107–114, doi:10.1109/FMCAD.2014.6987602.

[17] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur & Leander Tentrup (2017): *The 4th Reactive Synthesis Competition (SYNTCOMP 2017): Benchmarks, Participants & Results*. In: *Proceedings of SYNT@CAV*, *EPTCS* 260, pp. 116–143, doi:10.4204/EPTCS.260.10.

[18] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup & Adam Walker (2017): *The first reactive synthesis competition (SYNTCOMP 2014)*. *STTT* 19(3), pp. 367–390, doi:10.1007/s10009-016-0416-3.

[19] Swen Jacobs, Roderick Bloem, Romain Brenguier, Ayrat Khalimov, Felix Klein, Robert Könighofer, Jens Kreber, Alexander Legg, Nina Narodytska, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup & Adam Walker (2016): *The 3rd Reactive Synthesis Competition (SYNTCOMP 2016): Benchmarks, Participants & Results*. In: *Proceedings of SYNT@CAV*, *EPTCS* 229, pp. 149–177, doi:10.4204/EPTCS.229.12.

[20] Mikolás Janota (2018): *Circuit-Based Search Space Pruning in QBF*. In: *Proceedings of SAT*, *LNCS* 10929, Springer, pp. 187–198, doi:10.1007/978-3-319-94144-8_12.

[21] Mikolás Janota (2018): *Towards Generalization in QBF Solving via Machine Learning*. In: *Proceedings of AAAI*, AAAI Press.

[22] Mikolás Janota, William Klieber, Joao Marques-Silva & Edmund M. Clarke (2016): *Solving QBF with counterexample guided refinement*. *Artif. Intell.* 234, pp. 1–25, doi:10.1016/j.artint.2016.01.004.

[23] Mikolás Janota & Joao Marques-Silva (2015): *Solving QBF by Clause Selection*. In: *Proceedings of IJCAI*, AAAI Press, pp. 325–331.

[24] Mikolás Janota & Joao Marques-Silva (2017): *An Achilles' Heel of Term-Resolution*. In: *Proceedings of EPIA*, *LNCS* 10423, Springer, pp. 670–680, doi:10.1007/978-3-319-65340-2_55.

[25] William Klieber, Samir Sapra, Sicun Gao & Edmund M. Clarke (2010): *A Non-prenex, Non-clausal QBF Solver with Game-State Learning*. In: *Proceedings of SAT*, *LNCS* 6175, Springer, pp. 128–142, doi:10.1007/978-3-642-14186-7_12.

[26] Florian Lonsing & Armin Biere (2008): *Nenofex: Expanding NNF for QBF Solving*. In: *Proceedings of SAT*, *LNCS* 4996, Springer, pp. 196–210, doi:10.1007/978-3-540-79719-7_19.

[27] Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl & Armin Biere (2012): *Resolution-Based Certificate Extraction for QBF*. In: *Proceedings of SAT*, *LNCS* 7317, Springer, pp. 430–435, doi:10.1007/978-3-642-31612-8_33.

[28] Tomás Peitl, Friedrich Slivovsky & Stefan Szeider (2017): *Dependency Learning for QBF*. In: *Proceedings of SAT*, *LNCS* 10491, Springer, pp. 298–313, doi:10.1007/978-3-319-66263-3_19.

[29] Florian Pigorsch & Christoph Scholl (2009): *Exploiting structure in an AIG based QBF solver*. In: *Proceedings of DATE*, IEEE, pp. 1596–1601, doi:10.1109/DATE.2009.5090919.

[30] David A. Plaisted & Steven Greenbaum (1986): *A Structure-Preserving Clause Form Translation*. *J. Symb. Comput.* 2(3), pp. 293–304, doi:10.1016/S0747-7171(86)80028-1.

[31] Luca Pulina (2016): *The Ninth QBF Solvers Evaluation - Preliminary Report*. In: *Proceedings of QBF@SAT*, *CEUR Workshop Proceedings* 1719, CEUR-WS.org, pp. 1–13.

[32] Markus N. Rabe & Leander Tentrup (2015): *CAQE: A Certifying QBF Solver*. In: *Proceedings of FMCAD*, IEEE, pp. 136–143.

[33] Mate Soos, Karsten Nohl & Claude Castelluccia (2009): *Extending SAT Solvers to Cryptographic Problems*. In: *Proceedings of SAT*, *LNCS* 5584, Springer, pp. 244–257, doi:10.1007/978-3-642-02777-2_24.

[34] Leander Tentrup (2016): *Non-prenex QBF Solving Using Abstraction*. In: *Proceedings of SAT*, *LNCS* 9710, Springer, pp. 393–401, doi:10.1007/978-3-319-40970-2_24.

[35] Leander Tentrup (2017): *On Expansion and Resolution in CEGAR Based QBF Solving*. In: *Proceedings of CAV*, *LNCS* 10427, Springer, pp. 475–494, doi:10.1007/978-3-319-63390-9_25.

[36] Kuan-Hua Tu, Tzu-Chien Hsu & Jie-Hong R. Jiang (2015): *QELL: QBF Reasoning with Extended Clause Learning and Levelized SAT Solving*. In: *Proceedings of SAT*, *LNCS* 9340, Springer, pp. 343–359, doi:10.1007/978-3-319-24318-4_25.